A Research Program in Computer Technology

University of Southern California

July 1976

USC/INFORMATION SCIENCES INSTITUTE 4676 Admiralty Way Marina del Rey California 90291

ARPA ORDER NO. 2223

ISI SR 76-6

A RESEARCH PROGRAM IN COMPUTER TECHNOLOGY

Annual Technical Report
July 1975 - June 1976

prepared for the

Defense Advanced Research Projects Agency

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER ISI/SR-76-6	2 GOVT ACCESSION NO.	3 RECIPIENT'S CATALOG NUMBER
4 TITLE (and Subtitle) A Research Program in Computer Technology, Annual Technical Report, July 1975-June 1976.		5 TYPE OF REPORT & PERIOD COVERED Annual Technical Report July 1975-June 1976 6. PERFORMING ORG. REPORT NUMBER
7 AUTHOR(s) ISI Research staff		8 CONTRACT OR GRANT NUMBER(s) DAHC 15 72 C 0308
9 PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223 Program Code 3D30 & 3P10
11 CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12 REPORT DATE July 1976 13. NUMBER OF PAGES 98
14 MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ---		15 SECURITY CLASS. (of this report) Unclassified 15a DECLASSIFICATION DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution unlimited.

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

---

18 SUPPLEMENTARY NOTES

---

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

1. abstract data type, abstraction and representation, Alphard, Euclid, interactive theorem proving, lemma generator, Pascal, program correctness, program verification, software specification, verification condition
2. ARPANET, control memory, emulators, microprogramming, microprogramming language, microvisor, MLP-900, National Software Works, operating (OVER)

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes the research performed under Contract DAHC 15 72 C 0308 by USC/Information Sciences Institute from 1 July 1975 to 30 June 1976. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact.

(OVER)

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 102-014-6601

PRICES SUBJECT TO CHANGE

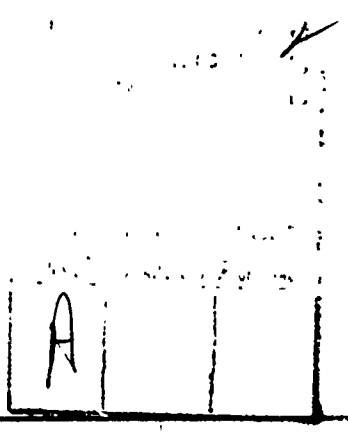
UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (continued)

2: systems, resource sharing, TENEX, time sharing, writable control memory.
3: abstract programming, domain-independent interactive system, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process transformation.
4: access control, computer security, encapsulation, error analysis, error-driven evaluation, error patterns, evaluation methods, protection mechanisms, software security.
5: computer terminals, interactive message service, office automation, nonprofessional computer users, terminal-based message service.
6: computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding.
7: distributed computation system, document printing capability, National Software Works, networks, network terminal, text printing, Xerox Graphics Printer.
8: ARPANET interface, computer network, KA/KI, KL2040, PDP-10, PDP-11/40, resource allocation, TENEX, TOPS20, user quotas.

20. ABSTRACT (continued)

The ISI program consists of eight research areas: Program Verification -- logical proof of program validity; Programming Research Instrument -- development of a major time-shared microprogramming facility; Specification Acquisition From Experts -- the study of acquiring and using problem knowledge for making informal program specifications more precise; Protection Analysis -- methods of assessing the viability of security mechanisms of operating systems; Information Automation -- development of a user-oriented message service for large-scale military requirements; Network Secure Communication -- work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; Special Projects -- a variety of activities and hardware developments in support of Institute programs; and ARPANET TENEX Service -- operation of TENEX service and continuing development of advanced support equipment.



USC/INFORMATION SCIENCES INSTITUTE 4676 Admiralty Way Marina del Rey California 90291

ARPA ORDER NO. 2223

ISI/SR-76-6

A RESEARCH PROGRAM IN COMPUTER TECHNOLOGY

Annual Technical Report
July 1975 - June 1976

prepared for the

Defense Advanced Research Projects Agency

Effective date of contract:	17 May 1972
Contract expiration date:	30 September 1978
Amount of contract:	$17,996,485
Principal Investigator and Director:	Keith W. Uncapher (213) 822-1511
Deputy Director:	Thomas O. Ellis (213) 822-1511

This research is supported by the Defense Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223, Program Code No. 3D30 and 3P10.

UNIVERSITY OF SOUTHERN CALIFORNIA

RESEARCH & ADMINISTRATIVE SUPPORT

Institute Administration:
Robert H. Blechen
Katherine Colegrove
Judy Gustafson
Georgene Petri

Librarian:
Rose Kattlove

Publications Group:
Nancy Bryan
Katherine Colegrove
G. Nelson Lucas

Secretaries to Directors:
Jeannette Christensen
Patricia A. Craig

CONTENTS

SUMMARY

This report summarizes the research by USC/Information Sciences Institute from 1 July 1975 to 30 June 1976. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact.

The ISI program consists of eight research areas: *Program Verification*--logical proof of program validity; *Programming Research Instrument*--development of a major time-shared microprogramming facility; *Specification Acquisition From Experts*--the study of acquiring and using problem knowledge for making informal program specifications more precise; *Protection Analysis*--methods of assessing the viability of security mechanisms of operating systems; *Information Automation*--development of a user-oriented message service for large-scale military requirements; *Network Secure Communication*--work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; *Special Projects*--a variety of activities and hardware developments in support of Institute programs; and *ARPANET TENEX Service*--operation of TENEX service and continuing development of advanced support equipment.

EXECUTIVE OVERVIEW

The Information Sciences Institute (ISI), a research unit of the University of Southern California's School of Engineering, was formed in May 1972 to perform research in the fields of computer and communications sciences with an emphasis on systems and applications.

A close relationship is maintained with USC academic programs through active cooperation among the Institute, the School of Engineering, the Department of Electrical Engineering, and the Computer Science Department. Ph.D. thesis supervision is an integral part of ISI programs, as is active participation of research assistants supporting ISI projects. ISI staff members frequently direct or participate in nationwide and international meetings and conferences; the institute also hosts frequent colloquia and seminars as a forum for distinguished speakers from other organizations.

The character and uniqueness of ISI are expressed in the following objectives:

- A major university-based computer science research center.
- A center with a largely full-time staff of researchers, augmented by graduate students and faculty.
- A center which possesses a unique blend of basic research talent and application and system expertise. The last two attributes are of special significance to the application of computer science and technology to key military problems.
- A university-based research center with strong active ties to the U.S. military community and a strong leadership role in identifying key computer R&D requirements in support of long-term military needs.

The Institute is structured to provide research and development capability at the system level--often required to assure an understanding of real problems and to provide useful solutions in transferable form. Project leaders share visibly in the responsibility for the conduct of each project and for the quality and impact of the research. At the end of the fourth year of operation, the full-time professional research staff numbers 43. The total number of ISI employees--including full-time research staff, participating faculty and graduate students, and support personnel--is 85.

The activities of ISI's eight major areas of research and associated support projects are summarized briefly below. Some of the research projects reported in this document are discrete activities in themselves; others can be seen as parts of a larger whole. For example, Program Verification, Specification Acquisition, and the Programming Research Instrument projects should be considered as individual parts of an overall research effort in Programming Methodology; Information Automation, Network Secure Communicatic ı, and Special Projects are linked elements of a major investigation into Network Communications Technology. These mutual interdependencies among the various projects at ISI contribute largely to the fruitfulness of the Institute's research activities.

Program Verification. The goal of program verification research at ISI is to develop an effective program verification system for proving that computer programs are consistent with precisely stated detailed specifications of what the programs are intended to do. The system is expected to replace significant parts of testing in current software development, and will also provide important tools for developing and judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques. Already running at ISI is an initial, experimental version of an interactive program verification system whose design philosophy is to provide automatic assistance for the verification process where practical, and otherwise to rely on human interaction. The system has verified numerous example programs. Important progress has been made in the following areas: improved user environment and interface to the verifier, extensible verification generator, algebraic approach to data abstractions including their verification, and influence of verification on language design. The eventual impact will be an increase in the quality of software.

Programming Research Instrument. PRIM is an interactive microprogrammable environment used for the emulation of existing or newly specified computer systems with major emphasis on providing debugging tools. These tools, available via NSW, provide the users with more powerful debugging facilities than available in the original target computer systems. The facility consists of a powerful microprogrammable computer (MLP-900), closely coupled to a TENEX operating system, and software to permit users to create and debug new emulators and target systems. Two prototype emulators, the UYK-20 and the U1050, have been completed and have been integrated into NSW. PRIM is an attempt to generalize a solution to the problem of software development through the use of emulation tools.

Specification Acquisition From Experts. The major effort of the SAFE project is simply to allow users who are not computer programmers to functionally specify their application directly to a computer system, with the system transforming this input into a precise functional specification of the application. This system is intended to be both independent of any particular problem domain and able to deal with "loose" (i.e., incomplete, inconsistent, etc.) problem-oriented descriptions of a domain through a dialogue with the user. From this dialogue the system can acquire the "physics" (the objects, laws, relationships, etc.) of the loosely-defined domain, structure it, and use it to

understand further communication and finally to rewrite the specification in precise operational form. The system is being developed in the context of a simplified real-world military specifications manual. An informal, incomplete specification for first-level message distribution has successfully been converted to a precise operational form. The system is now being expanded to deal with more complex specifications.

Protection Analysis. The goal of this project is to develop efficient techniques and semi-automated tools for detecting in operating systems various types of protection errors, i.e., errors that allow the systems to be compromised. The approach is empirical, based on the observations that (1) protection errors fall into a limited number of distinct classes and (2) techniques can be developed for finding the errors associated with each class. The method is to collect a data base of known errors, use it to determine the error classes, and (for each class) identify the relevant error characteristics for the purpose of developing an effective search algorithm. To date, errors from a variety of systems have been collected and techniques for finding errors for three of the classes have been reported. The project proposes to analyze and report on the remaining seven error classes.

Information Automation. The Information Automation project has a dual goal: 1) to develop the technology for providing on-line computer services directly to untrained users and 2) to develop a secure, on-line, interactive writer-to-reader message service for the military community. Such an on-line message service, new to the military, provides interactive assistance for formal messages from the initial draft preparation through coordination, transmission, and distribution. In addition, it will provide informal secure "off-the-record" communication to reduce the need for face-to-face meetings. During the past year the IA message service has progressed from a design on paper to a near-operational system. It is to be put at CINCPAC Headquarters on Ohau for formal testing in an operational environment, beginning in July 1977.

Network Secure Communication. The major objective of ARPA's Network Secure Communication project is to develop secure, high-quality, low-bandwidth, real-time, two-way digital voice communication over packet-switched computer communication networks. This kind of communication is a very high priority military goal for all levels of command and control activities. ISI's role in this effort is to develop efficient user-oriented systems for digital voice communications, primarily over packet-switched networks such as the ARPANET but also locally. The ISI NSC project is working on network voice protocols, digital voice conferencing systems, voice-oriented network host operating systems, real-time signal processing, hardware development, and other areas. During the past year (1) the Network Voice Conferencing Protocol (NVCP) was developed, (2) a sophisticated local CVSD conferencing system was implemented, (3) a signal-processor-based demonstration system was specified, proposed, acquired, and is being tested, and (4) the ELF operating system for the PDP-11 was extensively revamped and augmented to form the EPOS operating system.

Special Projects. The major efforts for the current year were as follows. First, further development was carried out on ISI's and ARPA's Xerox Graphics Printer (XGP), a high-quality printing capability in the form of a network terminal. Second, ISI reviewed technical progress and provided information and consultant service for the National Software Works, an ARPANET-based distributed operating system that is intended to provide a uniform computing environment for software developers. The third area is that of providing good human engineering for the military message service being developed by the Information Automation project. To this end, ISI is writing firmware to be used in a modified Hewlett-Packard 2640A terminal. The goal is to provide nearly instantaneous feedback for simple editing functions and flexibility by means of dynamic computer control of the range of available functions.

ARPANET TENEX Service. ISI is supporting, operating, and maintaining three complete TENEX systems on a schedule of 161 hours per week each, in order both to provide TENEX service to ARPA and to support its research projects via the facilities at ISI. The Institute provides 24-hour availability of TENEX systems, maintenance, and operators, continued development/improvement support, support of the XGP at IPTO, as well as ARPA NLS user support and minimal NLS software support. Through this support we have achieved increased long-term up-time, faster repair and improved preventive maintenance, economy of scale in operation, and the benefits of ISI expertise in establishing requirements for optimal loading and high reliability. In addition, this experience is used to assist in improving system reliability and to increase the number of users which can be handled with required response time.

1.
PROGRAM VERIFICATION

Research Staff:
Ralph L. London
Raymond L. Bates
David R. Musser
David S. Wile
Martin D. Yonke

Research Assistants:
Donald S. Lynn
Mark S. Moriconi
David G. Taylor

Consultant:
Lawrence M. Fagan

Support Staff:
Betty Randall

GOALS AND IMPACT OF PROGRAM VERIFICATION

In many computer application areas the consequences of a program not performing as intended can be quite costly or damaging. The goal of program verification research at ISI is to develop a prototype program verification system for proving that programs are consistent with precisely stated detailed specifications. With such a system one will be able to achieve significant confidence that computer programs will perform as intended. This system will be an important part of finding solutions to the manifest problems of current software systems--their high cost, their unreliable behavior, the difficulty of modifying them, etc. [Goldberg73]. The system will be used to help certify that software is correct; it is expected to replace significant parts of testing in current software development. It may be used in some cases to help determine whether protection and security specifications are met. The immediate impact will be that at last programmers will be able to demonstrate that their programs meet specifications. The system will also provide important tools for developing and judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques. The eventual result of advances in program verification will be an increase in the quality of software.

Last year's ISI Annual Report [AR75] contains a description of the existing verification system, named XIVUS, including an example of its use. During the current year we have demonstrated important progress in the following areas:

- Improved user environment and interface to the verifier.
- Extensible verification condition generator.
- Algebraic approach to data abstractions, including their verification.
- Impact of verification on programming language design.

Each of these achievements contributes to the overall goal of producing an effective interactive system for verifying significant programs that are written in several languages and that use current structuring and decomposition techniques. The results in each of the four areas are described sequentially below.

IMPROVED USER ENVIRONMENT

Program verification is a complex process. Consequently, part of the current verification effort is aimed at providing an environment which is helpful to the user whose goal is to verify a set of programs representing the solution of a particular task. During this year, three modifications were made to the verification system towards providing the desired environment.

1. A new "toplevel," called the EXECUTIVE, was designed and implemented. It provides a new command structure which guides the user through the verification process.

2. A user profile package was installed so that each individual may "tune" the system to his ideas of how the verification process is best performed.

3. The theorem proving component has been modified; it has a new, simpler command structure and records information used as the basis for any proved theorem.

The additions to the system are further explicated below, including the specific motivations for each addition.

New EXECUTIVE

The new EXECUTIVE was designed to ease the user through the verification process (a more detailed description, including an extensive transcript, can be found in [Yonke76]). Several specific factors were considered in its design:

- Only operations which make sense in the current state of verification are available to the user. For example, if the verification conditions have been generated for all the functions and procedures of the problem, then the "GENERATE verification conditions" command does not make sense. This implies that the available commands dynamically change based on the current state of verification.

- Item one should also apply to parameters or subcommands of the system. For example, the PROVE command should accept, as an item to prove, only verification conditions not already proved.

- The user of the XIVUS verification system should not be frustrated by giving an erroneous command and having the system respond with a "do not understand" message.

- The system should always have a reasonable suggestion for the next verification step until the entire verification process is complete. Causing the current suggestion to be performed should be very simple without impeding in any way the user's invoking any other reasonable operation to be performed.

- The user should have, at any point within the command structure, a simple way to interrogate the system for a list of the next alternative commands or parameters.

These goals were achieved by a command structure which exhibits the following characteristics:

- Command completion is similar to TENEX EXEC command completion. That is, if enough characters have been typed to disambiguate the command, then either typing the escape key (*esc*) or the space bar (*sp*) will be sufficient. Typing *esc* completes the command, including "noise words," while *sp* does not. The difference between this and the TENEX EXEC is that the system will not let you type in an "illegal" character, while TENEX will.

- There is one exception to the completion algorithm described above. Verification condition names are essentially the program name used to generate them and a number (used to identify different verification conditions within a program). When the system is expecting the user to input a verification condition name, after the user has typed a sufficient number of characters to identify the program name, the rest of the name is automatically printed out to the point where a number is needed. The user can then type the verification condition number. This eliminates the need for excessive typing.

- The command structure is tree-like in the sense that some commands take subcommands, which in turn might take subcommands. For example, the PRINT command is used to print programs, verification conditions, and verification status.

- At any point in the command structure, as mentioned above, the user may type a question mark (*?*) and the list of the next alternatives is printed out. In the example above, if the user typed a *?* after he had specified that he wanted to print a verification condition, a list of the verification conditions would be printed.

- Also, at any point in the command structure, the user may back up one level by typing a special key. He may then reselect from the menu presented. This may be repeated; therefore, after typing this special key a sufficient number of times, the user will be back at the top of the command structure.

- The space bar (*sp*) serves a special purpose at the beginning of a command or subcommand. If typed, the system suggestion is automatically taken and the command is typed out instead of the space. Therefore, if the user always wanted to take the system's suggestion, all he need do is hit the space bar.

The new EXECUTIVE greatly increases the ease of using the verification system. It should also be helpful in bringing new users up to competence quickly.

User Profile Package

The user profile package was motivated by two basic needs. One was to bring together, in a cohesive package, all the internal flags already in the system. These flags were known by the implementors, but this knowledge was hard to pass on. The other motivation was to have certain information about how each individual user wanted the system to perform for him. This information was incorporated into the system. The user profile package initiates an English dialogue with the user, asking him the appropriate questions. After the user response is interpreted, the internal flags are set. This package should be very helpful for a new user; since he can enter the dialogue at any time, he can change the system as he progresses in using the verification system.

Theorem Prover Modifications

Two modifications were made to the theorem prover. The first was to design a new command structure, similiar to that of the EXECUTIVE. During this process, new commands were chosen to (1) make the commands reflect the operations performed and (2) combine into one command operations which previously had to be done in a particular sequence. See [Yonke76] for a more complete description. The second addition was motivated by the fact that the user had to remember what he used for the basis of the proof of a particular theorem. The theorem prover now sends information to the EXECUTIVE, which remembers and can report which lemmas and rewrite rules--the basis for the proof--were used to prove a particular theorem.

This work on the verification system has made it much more usable, for both veteran and novice users. It helps the user by both guiding him through the verification process and keeping track of information which was formerly the user's reponsiblity. Even though it is possible to live without the new facilities, we insist on them for the analogous reason we insist on high-level programming languages rather than assembly codes. Further work is being done to provide new facilities to alleviate other user burdens.

EXTENSIBLE VERIFICATION CONDITION GENERATOR

The only component of a program verification system which is entirely new to computer technology is the "verification condition generator"--that portion of the system which determines what "theorems" must be proved in order to establish that a program does indeed have the specified properties. The XIVUS system is intended to be a tool for verification of programs written in several different languages. At the same time, research is under way into the best way to formulate and describe the rules for generating verification conditions for constructs in each of the languages.

The current verification condition generator is inadequate for these purposes in that it deals with only one language (a Pascal subset), and modification of it to incorporate better formulations of verification conditions is quite difficult. These and several additional considerations led to the design and implementation of a new verification condition generator which:

- Can be used with several different programming languages.
- Can be modified and extended easily.
- Will interface with an editor and parser to provide incremental verification condition generation (reproving a program with minor changes should be easy).
- Accepts an enriched "assertion language" used by the programmer to indicate the intended properties of the program.

Both the new and old verification condition generators are based on the ideas originated by King and Floyd, reformalized by Hoare, and developed by many others. However, several new ideas are incorporated in the new one. An example best illustrates the techniques. An inventory program which removes the "current order quantity" from the "on hand quantity" should probably have the property that afterwards the "on hand quantity" > 0. If there is an assignment statement in the program ONHAND ← ONHAND - CURRENTORDER, then a verification condition generator that "works backwards" will insist that the user prove that (ONHAND - CURRENTORDER) > 0. This "substitution rule" (right-hand side of the assignment for the left-hand side in the property to be proved) is actually quite dependent on the forms of the data structures involved. For example, ONHAND might actually be a file access parameterized by "part number." For this reason, the new verification condition generator does not make the substitution above; it merely indicates that one is to be made. The programming language designer must provide the substitution rules to be used. In general, the new generator never makes a decision that is either language-dependent or more properly made by another component of the verification system.

To continue the example, the programmer might precede the above assignment with a test: if CURRENTORDER > ONHAND then goto BACKORDER. A verification condition generator which "works forward" will conclude that after this statement whatever was true before the if still holds. In addition, ~ (CURRENTORDER > ONHAND) will hold. A verification condition that could be generated is

```
~ (CURRENTORDER > ONHAND)
  implies
    (ONHAND - CURRENTORDER) > 0
```

(which is not provable since it is not in fact quite true--there is an inconsistency between the given property and the program).

In general, for any path through a program the path can be "marked" and a verification condition generated that the predicate at the mark obtained by working forward implies that obtained by working backward. The new technology involved is to save both forward and backward predicates with each program node, enabling incremental verification and enhancing the options for verification condition generation: pure forward, pure backward, or a mixture are all possible within the new system.

The new system allows for different languages by providing an extension mechanism: new syntactic constructs may be defined for any particular programming language in terms of old ones. Verification conditions and/or Hoare axioms for the construct are produced from the definition by the generator. The researcher in verification technology or programming language design may then examine and modify these rules, which are automatically invoked when the syntactic construct is used in a program.

The basis for this technology--associating the predicates with the nodes--was developed in part by Gerhart [Gerhart75]. Although the technology here is innovative, the existence of the tool is more important. Future extensions to the verification condition generator include enriching the syntactic constructs allowed (to parallel constructs), enriching the semantics which can be conveyed to the generator, encapsulating the substitution rules in a language-independent manner, and linking the generator into the XIVUS system.

NOTE: The inconsistency can be removed by changing the given property to "on hand quantity ≥ 0."

ALGEBRAIC APPROACH TO DATA ABSTRACTIONS

In the algebraic approach to data abstractions, an abstract data type is defined to be a set of operations on a set of objects and a set of equations relating the operations. Regarded as *axioms*, the equations serve as a representation-independent *specification*, on which applications of the data type can be based and against which implementations can be compared. An *implementation* of a data type consists of a representation, which indicates how the abstract objects are to be represented in terms of some other data type(s), and set of programs for the operations expressed in terms of the representation. A *correct implementation* is one which satisfies the axioms of the specification.

One of the key ideas of the approach taken in [Guttag76a] is to express both the axioms of the specification and the programs of an implementation of a data type as *rewrite rules*. With a few relatively minor restrictions on their form, these rules can be compiled or interpreted by a relatively simple *pattern-match compiler* or interpreter. Such pattern-matching systems already exist in symbolic mathematical systems such as Reduce, Scratchpad, and Macsyma, and extensive use has been made of the Reduce pattern-match interpreter in the work reported in [Guttag76a] and [Guttag76b].

Expressing both axioms and programs as rewrite rules suggests a *duality* between specifications and implementations that has important consequences for software design, testing, and verification. In design of data types, the duality suggests that the task of initially axiomatizing a data type can be approached as a programming task, using *expressions* (operator-operand tree structures) as a representation. Using this approach, it has been relatively easy to axiomatize familiar data types such as stacks, queues, binary trees, strings, sets, lists, graphs, and files [Guttag76b], as well as newly invented types such as "message exchange."

Since the axioms of a data type can be compiled into running programs, this implementation can be used for initial testing with actual or symbolic data, permitting the designer to test to a limited extent whether his specification captures the properties intended. One can also test high-level algorithms which are programmed in terms of the data type before fixing upon an actual implementation of the data type. Thus, a true top-down design methodology can be achieved [Guttag76a].

Perhaps the most important consequences of the axiom/program duality are in verification. By using both axioms and programs as rewrite rules in proofs, the proofs become in large part very straightforward and computational in nature. In this respect the proof method is very similar to some of the methods of [Boyer75] for verifying Lisp programs. Combination of the axiomatic approach to data types and hierarchical development of software results in a very important advantage in verification, namely, factoring proofs into the same hierarchical structure as in the programs [Hoare72]. This "levels of abstraction" approach becomes particularly attractive with algebraic axioms because of the possibility of constructing axiom sets which are in an important sense *complete* [Guttag75].

An Example

The characteristics of algebraic axiom specifications and their use in verification are nicely illustrated by the example of a symbol table data abstraction and its implementation by a stack of hash tables. This example, first introduced in [Guttag75], provides a collection of operations for maintaining a symbol table, such as might be used in a compiler for a block structured language. The informal specifications for these operations include: (1) upon block entry one can redeclare previously used symbols (former attributes become inaccessible until exit from the block); and (2) upon exit from a block, attributes declared in the block become inaccessible. The formal specification with algebraic axioms defines six operations on symbol tables with nine equations which relate operations to each other, e.g., LEAVEBLOCK(ENTERBLOCK(symtab)) = symtab. The axioms define the behavior of the operations precisely, yet do not prescribe or preclude any particular implementation. The operations in an implementation are programmed in terms of operations of other data abstractions, for which there are axiomatic specifications which can be used in carrying out the verification that the symbol table axioms are satisfied. The symbol table operations can, for example, be implemented by a Stack of Mappings, the Stack being implemented by an array/index pair and a Mapping by a particular kind of hash table (an array containing lists of identifier/attribute pairs). The verification of the symbol table axioms then involves use of a set of axioms for Stack operations, e.g., POP(PUSH(stk,item)) = stk, and a set of axioms for Mappings, e.g.,

```
EVALMAP(DEFMAP(map,id,attr),id1) =
        if id=id1 then attr else EVALMAP(map,id1).
```

(The form of this axiom, with the conditional expression and recursive occurrence of EVALMAP on the right-hand side, is typical of the form used in algebraic axiom specifications.) The particular implementations chosen for Stacks and Mappings are verified in the same way, using basic sets of axioms for arrays and lists. It is important to

note that the verification of the top level of the symbol table implementation does not require knowledge of the particular implementations of Stacks or Mappings, only their axiomatic specifications. Thus the proof of the entire implementation factors nicely into levels.

Semi-Automatic Verification of Data Type Implementations

To carry out the verification of data type implementations using the algebraic approach, a prototype set of facilities has been added to the XIVUS system. Using these facilities, the complete implementation of the symbol table data type using a stack of hash tables has in fact been verified. As an example of the proof process, consider the verification of the top level implementation by a Stack of Mappings. The first step is to direct the system to adopt the programs of data type symbol table and the axioms of data types Stack and Mapping. These programs and axioms would all be in the form of rewrite rules which the user had just entered or had read in from files. The command for "adopting" a set of rules is separated from the act of reading them in so that several sets of rules for an operator can coexist within the system. Assuming that the symbol table axioms have also been input to the system, the user then directs the system to generate the verification conditions for the data type. These would consist of the symbol table axioms and the "equality axioms" for the symbol table equality operator, all interpreted in terms of the representation.

The user can then attempt to prove each of the verification conditions using CEVAL, a special "conditional evaluator" which has been developed primarily for this purpose [AR75], [Guttag76a], or the standard simplifier/theorem prover of the system. In these proofs the rewrite rules from the Symbol Table programs and Stack and Mapping axioms are used automatically, without further direction from the user. In some cases, completion of a proof requires one or more assumptions to be made about the representation or the Stack or Mapping data types. Initially these assumptions are input by the user and used as needed without justification. To complete the verification of the implementation, it is necessary to prove these assumptions, or a stronger set of assumptions, as theorems (about the symbol table data type implementation or about the Stack or Mapping data types). The verification conditions sufficient to establish these theorems are constructed using the domain/range specifications of the data types, in accordance with the principle of induction on the number of applications of operations of the data type [Hoare72].

Work is now in progress on extensions to the basic methodology of the algebraic axiom approach to permit operations with side effects and to incorporate error handling systematically. These extensions will contribute toward integrating the data abstraction components of the XIVUS system with the existing Pascal verification components and future components for verification of programs in other languages.

THE IMPACT OF VERIFICATION ON LANGUAGE DESIGN

In addition to verifying existing programs, written mainly in Pascal, we have been deeply involved this year in the design of two new programming languages, Euclid [Lampson76] and Alphard [Wulf76, Shaw76b]. Both of the language designs have as one of their important goals verifiability of the resulting programs. Naturally, additional goals and numerous other concerns are exerting major influences on these languages. Nevertheless, it has been both surprising and extremely pleasing to observe the degree to which these concerns have reinforced each other. We deem it quite appropriate to provide a short glimpse into the interactions, starting with Euclid.

The Euclid language, drawing heavily on Pascal and deliberately restricted to current knowledge of programming languages and compilers, is intended for the expression of system programs which are to be verified. Both the language and its compiler are given part of the task of producing a correct program and of verifying its correctness. For example, although global variables are permitted, they must be explicitly listed when used in a procedure or a record. This explicit listing means that no reader of a program need do computing or complex searching to determine the global variables. One class of readers in particular, human or mechanical verifiers, has this information readily available for use. Furthermore, the language is able to guarantee that two identifiers in the same scope can never refer to the same variable, i.e., there is no aliasing. All of this by deliberate design meshes well with a new, easily explained proof rule for verifying procedure definitions and calls. The proof rule, developed for Euclid from several existing proof rules, captures exactly the full Euclid procedure definition and call mechanism and also removes restrictions and known problems with other proof rules. In a very real sense, the Euclid design is one of adding restrictions and the enforcing mechanisms to meet a desired level of understandability and verification capability.

The use and verification of pointers in Euclid is made easier than in other languages by allowing each dynamic variable to be assigned to a language construct, the *collection*, and guaranteeing that two pointers into different collections can never refer to the same variable. Thus assertions need not be invented and verified to obtain this guarantee; instead it is all part of the language.

When possible, the above guarantees are provided by extensive compile-time checks. If the compiler is unable to complete a check, it generates *legality assertions* for the verifier to establish. Verification concepts are thereby used to complement other mechanisms.

The Alphard language is a new language design rather than one starting from an existing language. The effort focuses simultaneously on issues of programming structure (methodology) and verification. The abstraction mechanism of Alphard, the *form*, encapsulates a set of related function definitions and associated data descriptions, allowing a programmer to reveal the behavior of an abstraction to other users while hiding information and protecting details of the concrete implementation.

This explicit distinction between the abstract behavior of a data abstraction and the concrete program that happens to implement that behavior provides an ideal setting in

which to apply Hoare's techniques for proving the correctness of data representations [Hoare72]. In the Alphard adaptation one shows that the concrete representation is adequate to represent the abstract objects, that it is initialized properly, and that each operator provided on the abstract objects both preserves the integrity of the representation and does what it is claimed to do (in terms of both the abstract behavior and the concrete procedure that actually implements the operator).

The verification technique and the methodology decisions both require providing specifications of the abstract objects and the related operations. They also need conditions describing the concrete objects and operations, invariants holding over all operations, and a representation function giving the relation between concrete and abstract objects. All of this information, made an integral part of a form definition, was originally included for verification reasons. Its presence, however, has directed attention toward things which, on methodological grounds, ought to be of concern. The verification technique exposed the need for certain language features, which at best were viewed as conveniences and at worst would have been missed completely on the basis of methodological or language considerations alone.

Methodology concerns have also benefited verification. The entire form concept, for example, was introduced for methodological reasons. It is this factorization and isolation, however, which appears to make either hand or mechanical verification feasible. Similarly the notion of *generators*, which permits hiding certain details of iteration, was introduced on methodological grounds, but is also simplifying the verification of many loops. Loop control using generators is implicit rather than explicit, and therefore a single verification of that loop control suffices for all of its invocations.

An important part of language design is knowing what should be left out. During the Alphard design, constructs were repeatedly proposed which either gave difficulty in formulating the needed proof rules or which looked suspect on methodological grounds. Usually such a problem signalled an unforeseen problem in the other domain. For example, an early version of the iteration statement was much more elaborate than the one currently adopted. Nevertheless, it seemed plausible on methodological grounds. Its verification, however, was a horror to behold. Subsequently it became apparent that the complexity of its verification was symptomatic of a difficulty which any programmer would have in attempting to understand the statement or its use.

Numerous example forms, and programs using these forms, have been designed and verified [Wulf76, Shaw76b, London76, Shaw76a]. The proofs are modular, reflecting the structure of the programs. In addition, the lengths of the proofs are within reasonable limits and indeed quite encouraging. Most importantly, when modifications have been made to a program, corresponding modifications needed in the proof have been nearly always easy to identify and to complete, without the need to redo the entire proof. If the implementation of an abstraction is changed, but not the specifications, then all programs using the abstraction and all verifications of those uses are also unchanged.

Even if we never verify another program (and no one even remotely believes that to be the case), already the impact of verification on language design and the expression of quality programs is significant and worthwhile. In fact, one of our colleagues,

responding to an early revision of one of these languages, noted that it is "thrilling to see verification finally interacting with language design."

REFERENCES

[Boyer75] Boyer, R. S., and J S. Moore, "Proving Theorems about LISP Functions," *J. ACM*, Vol. 22, No. 1, January 1975, pp. 129-144.

[Gerhart75] Gerhart, S. L., "Correctness-Preserving Program Transformations," *Conference Record of the Second ACM Symposium on Principles of Programming Languages*, 1975, pp. 54-66.

[Goldberg73] Goldberg, J. (ed.), *Proceedings of a Symposium on the High Cost of Software*, Monterey, California, September 1973. Published by Stanford Research Institute.

[Guttag75] Guttag, J. V., *The Specification and Application to Programming of Abstract Data Types*, Ph.D. thesis, University of Toronto, Department of Computer Science, Computer Systems Research Group Technical Report CSRG-59, 1975.

[Guttag76a] Guttag, J. V., E. Horowitz, and D. R. Musser, *Abstract Data Types and Software Validation*, USC/Information Sciences Institute, 1976.

[Guttag76b] Guttag, J. V., E. Horowitz, and D. R. Musser, *The Design of Data Structure Specifications*, USC/Information Sciences Institute, 1976.

[Hoare72] Hoare, C.A.R., "Proof of Correctness of Data Representations," *Acta Informatica*, Vol. 1, No. 4, 1972, pp. 271-281.

[Lampson76] Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchell, and G. J. Popek, *Euclid Report* (draft), 1976.

[London76] London, R. L., M. Shaw, and W. A. Wulf, *Abstraction and Verification in Alphard: A Symbol Table Example*, Carnegie-Mellon University and USC/Information Sciences Institute, 1976.

[AR75] *A Research Program in Computer Technology: Annual Technical Report, May 1974 - June 1975*, USC/Information Sciences Institute, ISI/SR-75-3, September 1975.

[Shaw76a] Shaw, M., *Abstraction and Verification in Alphard: Design and Verification of a Tree Handler*, Carnegie-Mellon University, 1976.

[Shaw76b] Shaw, M., W. A. Wulf, and R. L. London, *Abstraction and Verification in Alphard: Iteration and Generators*, Carnegie-Mellon University and USC/Information Sciences Institute, 1976.

[Wulf76] Wulf, W. A., R. L. London, and M. Shaw, *Abstraction and Verification in Alphard: Introduction to Language and Methodology*, Carnegie-Mellon University and USC/Information Sciences Institute, 1976.

[Yonke76] Yonke, M. D., *The XIVUS Environment*, XIVUS Working Paper No. 1, USC/Information Sciences Institute, April 1976.

2.
PROGRAMMING RESEARCH INSTRUMENT

Research Staff:
Louis Gallenson
Alvin Cooperband
Ronald Currier
Joel Goldberg
Raymond L. Mason

Research Assistant:
Ben Britt

Support Staff:
Rennie Simpson

INTRODUCTION

PRIM is an interactive microprogrammable environment used for creating emulators of existing or newly specified computers with major emphasis on providing programming debugging tools; it is available via remote terminals through the ARPANET. PRIM provides editors, compilers, and debuggers for creating emulators as well as an environment for providing target systems with debuggers and configurors that can be operated by the user in the familiar language of the original system. The emulated machine generally provides better user debugging facilities and greater flexibility in system configuration than the original machine, while producing bit-to-bit compatible results on all levels of execution. PRIM is an attempt to generalize a solution to the problem of software development by means of emulation tools; it is a unique and powerful facility for improving software development within the DoD user community.

The goals of the PRIM project are to facilitate more efficient programming by providing and demonstrating integrated emulation-based tools that can give the user the ability to create, debug, and execute programs for target machines in an interactive (time-shared) environment richer in necessary user facilities than the original. These tools are integrated into the National Software Works (NSW) system (see Section 7), making them available on the ARPANET. PRIM is therefore a service facility, providing unique tools to NSW programmers, as well as an experimental computer environment for the researcher. The major implementations are the PRIM environment, a tool for emulator tool builders, and two sample emulations, a UYK-20 tool and a U1050 tool.

The use of emulations (i.e., simulations) of unavailable computer systems as an aid in programming large systems is certainly not novel. The uniqueness of PRIM is to provide an integrated set of user tools, available via remote terminals, utilizing a well supported general-purpose multiaccessed computer system (TENEX) and near real-time emulations, rich in debugging aids, for software development. Like NSW, the major aim of the PRIM project is to permit the military community easy access to the most recent computer

technology, allowing the use of an existing operating system, editors, compilers, and other programs for creating and debugging new software. In addition, PRIM is demonstrating the ease of introducing some types of new tools into NSW by emulating the computer system rather than implementing hardware and software compatible with the protocols of the NSW operating system and the ARPANET.

The major project effort for this reporting period is the design and implementation of a PRIM system tailored to the needs of military programmer while enhancing the emulator-writing capabilities for additional tools operable within the PRIM facility. This new effort utilizes the PRIM facility completed in 1975. (A detailed description is found in the *PRIM User's Manual* and need not be repeated here.) The tasks being completed for this reporting period are as follows:

- New TENEX MLP-900 Driver
- PRIM Exec
- PRIM Debugger
- PRIM Tool
- UYK-20 Tool
- U1050 Tool

THE PRIM FACILITY

The PRIM system was developed at ISI as a subsystem of TENEX. PRIM consists of an MLP-900 microprogrammable processor and appropriate software to drive the MLP-900, to support MLP-900 microprogramming, and to provide an environment in which users create, manipulate, and interact with their emulators and/or emulated systems.

Hardware

PRIM's hardware system is based on two processors: the shared use of a Digital Equipment Corporation's PDP-10 with other network users and the STANDARD Computer Corporation's MLP-900 prototype processor. The PDP-10 and MLP-900 share memory as dual processors; the MLP-900 is also a device on the PDP-10 I/O bus. The PDP-10, connected to the ARPANET, runs under TENEX with a paged virtual memory. Its processor contains 256K words of 36-bit memory. The I/O operations performed by TENEX include file, terminal, and network handling, swapping, and all other accesses to peripheral devices.

The MLP-900 is a fast, powerful vertical-word microprogrammed computer that has been tailored to interface the TENEX system. It contains 4K 36-bit words of control memory, 80-nanosecond cycle time, and runs asynchronously with a 4 MHz clock. A major modification of the MLP-900 has been the introduction of a supervisor state which allows

the processor to be shared with full protection between users. Prior to this project, little had been done toward making the multitude of available microprogrammed processors potentially sharable resources. This initial experiment goes a long way toward making microprogrammed processors widely and inexpensively available. The hardware environment was completed in 1974.

Software

The principal items of PRIM software are the General Purpose Microprogramming Language (GPM) compiler, the MLP-900 microprogram supervisor (microvisor) and the MLP-EXEC. The remaining software -- TENEX MLP-900 Driver, PRIM Exec and Debugger (PRIM Tool), the UYK-20 emulator, and the U1050 emulator -- was implemented during this reporting period.

GPM and the GPM Compiler. GPM is a high-level machine-oriented language, designed explicitly for writing programs for the MLP-900. As a high-level language, GPM offers a block structure and statement syntax similar to PL/1 or ALGOL. The compiler is capable of producing multi-instruction code per statement as well as statements producing exactly one MLP-900 instruction per statement. The GPM compiler was essentially completed in early 1974; for a more detailed account of its development the reader should consult the *PRIM User's Manual*.

MLP-900 Microvisor. The MLP-900 microprogram supervisor (microvisor) is a small, fully protected resident system that controls the MLP-900 and its communication with the PDP-10. It loads and unloads the user's MLP-900 context upon command from the PDP-10, supports paging of the user target program, protects main memory and the rest of the PDP-10 system from emulator errors, and provides the emulator with a few other services. The microvisor interacts only with the user microcode and the TENEX MLP driver

The TENEX MLP-900 Driver. Access to the MLP-900 from a TENEX process is accomplished via the MLP driver in TENEX. The driver is responsible for initializing the MLP-900 microcode, controlling and swapping users, and passing along all the I/O requests. The driver is an extension of the microvisor; all communication with the MLP-900 goes through the driver, while communication with the driver occurs through the normal I/O JSYS's.

To improve the security and efficiency of operation, the current version of the driver has been incorporated into the TENEX monitor. The new TENEX MLP-900 driver will appear functionally the same to PRIM users as the original driver (written as a user program) with improvements in responding to page faults, swapping, and I/O requests. The MLP-900 is an I/O device to TENEX and uses existing system calls for communication. The security of the system was improved by denying the PRIM users the ability to write

and load their own microvisor. This capability is restricted to maintenance mode and only used by ISI personnel.

PRIM Exec. The PRIM Exec has replaced MLP-EXEC for the emulator user. (MLP-EXEC, described in the *PRIM User's Manual*, served as the vehicle for creating and checking out MLP emulations prior to the development of the PRIM Exec and is replaced by the PRIM Exec.) The PRIM Exec provides the environment in TENEX needed to support each of the PRIM MLP-900 emulators, together with a command language allowing the user of a particular (emulated) computer to access that environment with the already familiar vocabulary of that computer.

The emulator support consists of the module responsible for controlling (emulated) execution, plus a server responsible for satisfying the emulator's I/O requests. This I/O server offers the emulator a full range of I/O operations, including "magnetic tape" and "disk" operations (actually performed on structured TENEX disk files).

A command language interpreter in the PRIM Exec provides a uniform terminal interface modelled after the TENEX Exec, but with commands oriented toward the needs--and vocabulary--of the programmer familiar primarily with the computer being emulated; additional facilities are provided the emulator-writer for the development and checkout of a new emulator for the PRIM system. (The tailoring of the PRIM Exec, and also the PRIM debugger, to the details of a particular emulated machine, including its terminology, is accomplished through a set of machine-specific tables that accompany each emulator.)

The majority of the commands concern the building (and interrogating) of the emulated machine's environment, e.g., installing devices on the machine, mounting TENEX files on those devices, and modifying the size of memory. In addition, there are commands that allow a complete checkpoint and subsequent restoration of the user's state and the generation of a transcript of all or part of a PRIM session.

PRIM Debugger. The PRIM debugger is a table-driven interactive symbolic debugger that permits a user of the PRIM system to debug target-machine programs in terms of symbols defined for the target machine, using the data representation and instruction formats of that machine. The debugger also provides symbolic access to the MLP context; it uses a command language with feedback and help available if needed.

Some of the abilities provided by the PRIM Debugger are

1. To evaluate expressions of arbitrary complexity using the target machine's arithmetic and recognizing the target machine's symbols.

2. To display and modify the contents of lists and ranges of target-machine location, including some control panel functions.

3. To set and clear any number of read, write, and execute program breaks (or any combination thereof) whenever in the target machine they are appropriate (breakpoints are marked in metabits in target memory).

4. To display a history of the most recent target program jumps and transfer control to designated addresses.

5. To assemble symbolic target machine instructions on input and disassemble them on output, including symbolic expressions as addresses.*

The tables that supply machine-specific information to the debugger are supplied by the emulator developer. Among the information contained are

1. Descriptions of the properties of various target-machine and MLP address spaces.

2. Symbol tables for all target-machine symbols and appropriate emulator symbols.

3. An instruction format description, including symbolic op codes.

4. Routines to convert from the internal data representation of the debugger to the data representation of the target machine and vice versa.

5. Routines to perform arithmetic and logical operations according to the conventions of the target machine.

DESCRIPTION OF PRIM TOOL

As a tool, PRIM is the hardware and software mentioned above, plus a new user's manual directed towards future emulator implementors; the latter is the only component lacking (completion is scheduled for FY77). In general, we are striving for a complete, exact emulation of the target machine, including not just instructions and registers, but also clocks, interrupts, machine states, memory protection and relocation, and nearly-real i/O. Complete instruction sets, functionally identical to the emulated CPU, are required to produce bit-compatible results in the working registers. The actual implementation of machine instructions is transparent to the user; the emulation need be "correct" only at those windows in the emulated cycle where interrupts may logically occur.

---

* Defined by target machine tables Scheduled for completion by October 1976.

Also, we are striving for a better target system for debugging new programs even at the expense of slower execution time. To achieve this goal, PRIM emulators are limited to 32-bit computers, where the extra bits are used as meta-bits for the conditional breaks. Emulated timing information is provided for instruction executors as memory references and I/O. Jump history queues are provided for several of the most recent target system jumps. A number of parameters are provided to allow users to easily configure and save individual computer systems (I/O devices, memory size, speed of CPU, etc.). The major attributes for PRIM-based tools is in its flexibility and in that the debugging environment is external to the target machine and does not interfere or change the run-time properties of the target programs.

AN/UYK-20 EMULATOR

We have completed a PRIM-based emulation of the AN/UYK-20 which provides a complete and accurate AN/UYK-20 processor, as detailed below; included in the emulation are CPU instruction execution, channel instruction execution and data transfer, clocks, interrupts, control panel switches, and an assortment of asynchronous I/O devices.

Instruction Execution

The complete basic instruction set is implemented; the optional instructions (MathPack) are not included in the initial release but will be available in FY77. Where the AN/UYK-20 specification states a restriction on the use of an instruction, but does not specify the consequences of violating that restriction (e.g., requiring even registers in "double" instructions), the emulator halts and reports a program anomaly when such a violation occurs. Indirect addressing under the control of Status Register 2 and relative addressing via page registers are included.

Memory

Both main memory and NDRO are implemented. Main memory size may be altered in 8K increments; the available memory is assumed to be contiguously addressable from zero. The memory is a single-port memory (but adding DMA as part of a device requiring it is straightforward).

IOC EXECUTION

I/O Controller (IOC) command and chaining execution are implemented, together with byte, word, and double-word data and function transfers. All sixteen channels, and their control memory, are available. All data transfers (between emulated channels and

emulated devices) are byte-parallel, using the natural byte size of the device. Data (and function) transfers in all cases are driven by the device, at the device's rate, ignoring any programmed modulation rate.

This implementation is correct for parallel and NTDS channels. For asynchronous communication channels, the only visible effect on programs is that the data transfers run "correctly" regardless of the serial information provided to the channel. For synchronous communication channels, an additional problem arises when there is, in fact, a pure bit stream; for such a device, the emulation must do bit-at-a-time transfers, using the serial info to reassemble the "bytes".

Clocks and Timing

Both real-time and monitor clock registers are implemented, with a user-specifiable modulation rate (the default is the internal rate, which is one millisecond). AN/UYK-20 time is counted in a 50-nanosecond internal timer; no relationship is specified between real (MLP) time and AN/UYK-20 time. Emulated instruction timing (both CPU and IOC) is counted according to the AN/UYK-20 specification.

Interrupts

The complete interrupt facility is implemented, and all interrupts (except power fault) are generated when appropriate. Power fault (and any other interrupt) can be forced by the user by setting the associated flag via the debugger.

Console Switches

Various control panel switches are implemented as cells which can be manipulated by the user (using the debugger). Normal toggles are set by the user and sensed by the emulator; return-to-neutral toggles are set by the user, then cleared by the emulator after it has sensed the setting. The implemented switches are Bootstrap, Load/Stop, Program Stop, and Clock Disable. There are no indicators as such; all the registers and state information are accessible through the debugger.

I/O Devices

All emulated I/O devices run asynchronously with respect to the AN/UYK-20 processor (CPU and IOC), scheduling themselves for MLP service in terms of the internal (50-nanosecond) timer. Device timing is based upon a single parameter which expresses the time required for one basic operation, typically the interbyte time interval. Scheduling

for all operations is based upon fixed relationships with the timing parameter. The parameter for each device can be set by the user, with the default giving the actual device speed.

The initial complement of devices, required by the Level II software, is as follows:

- Univac 1532 I/O console.
- Cipher Mark I magnetic tape system. Reads and writes emulated tape files within the TENEX disk system; a utility for converting between real magnetic tapes and these emulated tapes is available.
- Documation card reader. Capable of reading either TENEX ASCII text files or card binary files.
- Versatec matrix printer.

Channel and Device Configuration

Installation is done on a channel-by-channel basis. Each installed channel may have any implemented device installed; the emulation system has no restrictions regarding channel groups, and does not enforce any such restrictions. Installing a device implicitly specifies the type of channel; one may not mount a device on the wrong type of channel. Mounting is done on installed devices by associating TENEX file(s) with the device. For devices which are actually multidevice controllers, mounting is done separately on each unit. In general, a single TENEX file is mounted on each device; in the case of emulated terminals, however, separate input and output files are needed when disk files are to be used. Translation of data, where applicable, is specified at this time. The speed (transfer rate) of a device can be specified at any time; the default is the actual device speed.

U1050 TOOL

We have completed a PRIM-based emulation of the U1050 which provides a complete and accurate processor as detailed below; included in the emulation are CPU instruction execution, I/O instruction and data transfer, clocks, interrupts, control panel switches, and an assortment of asynchronous I/O devices.

Instruction Execution

The complete instruction set is implemented. As with the AN/UYK-20, where the U1050 specification states a restriction on the use of an instruction but does not specify the consequences of violating that restriction, the emulator halts and reports a program anomaly when such a violation occurs.

Memory

Main memory size may be altered in 8K-byte increments; the available memory is assumed to be contiguously addressable from zero.

Clocks and Timing

U1050 time is counted in a 50-nanosecond internal timer; no relationship is specified between real (MLP) time and U1050 time. Emulated instruction and controller timing is counted according to the U1050 specification.

Interrupts

The complete interrupt facility is implemented, and all interrupts (except internal parity errors) are generated when appropriate. Any interrupt can be forced by the user by setting the associated flag via the debugger.

Console Switches

Various control panel switches are implemented as cells that can be manipulated via the debugger. Normal toggles are set by the user and sensed by the emulator; momentary switches or toggles are set by the user, then cleared by the emulator after it has sensed the setting. The implemented switches are Clear, Start, Continue, Card-load, Tape-load, Operator request, and three sense switches. There are no indicators as such, since other control functions are available through the debugger.

I/O Devices

All I/O devices are implemented except for high-speed communications. All channels (except 4 and 5) are available. All data transfers (between emulated channels and emulated devices) are byte-parallel, using the natural byte size of the device. Data transfers in all cases are driven by the device, at the device's rate.

All emulated devices run asynchronously with respect to the U1050 processor, scheduling themselves for MLP service in terms of the internal (50-nanosecond) timer. Device timing is either fixed in the emulation or is based upon a single parameter that expresses the time required for one basic operation, typically the interbyte interval. Scheduling for all operation is based on fixed relationships with the timing parameter. The parameter for each device can be set by the user, with the default giving the actual device speed.

The implemented I/O devices consist of a printer, card reader, card punch, low-speed communications (up to 15 units), mass storage (one disk unit), and one read/write tape unit.

Device Configuration

Installation is done on a channel-by-channel basis. As each U1050 channel is committed to a particular type of device, care must be exercised that each device is installed on its proper channel.

CONCLUSIONS

By October 1976 the PRIM project will have realized all its major goals: to provide a rich microprogramming environment for computer scientists, provide programming tools under NSW to the military community, and make the technology available to the DoD community. (As mentioned above, the necessary documentation will be completed during FY77.) The PRIM facility will remain at ISI and continue to support the efforts of NSW and SDL in providing integrated computer tools for programmers and system designers. PRIM personnel will continue to support these users by completing the PRIM environment (adding configuror), completing the user documentation, providing user guidance, and improving the capabilities of the existing tools. Two planned implementations are a dual UYK-20 emulation and a controller for real-time inputs.

When these tools are completed, the PRIM project will have successfully completed the PRIM tool and the UYK-20 and U1050 tools. We are optimistic about user acceptance of these tools and the whole PRIM environment. The PRIM architecture (characterized by the dual processors, one of which is multiaccessed, available via remote terminals, and provides a well supported general-purpose programming environment rich in programming tools and the other of which is a fast microprogrammable CPU) is the correct approach to providing emulation-based programming tools. The military community has recently shown interest in using emulation-based tools to implement and develop large software systems. The existence and general availability of PRIM should provide incentives for the development of additional PRIM-like systems for general use.

REFERENCE

Gallenson, Louis, et al., *PRIM User's Manual*, USC/Information Sciences Institute, ISI/TM-75-1, April 1975.

3.

SPECIFICATION ACQUISITION FROM EXPERTS

Research Staff:
Robert M. Balzer
Neil M. Goldman
David S. Wile

Support Staff:
Nancy Dechter

INTRODUCTION

Only modest gains in programming productivity have been produced in 25 years of software research, but the groundwork has been laid for major advances through rationalization and automated aids. This groundwork rests on two critical ideas: that specification must be separated from implementation, and that the separation between these two processes should be a formal operational abstract (i.e., very high level) program rather than a nonoperational requirements specification. Structured programming represents the first results of combining these ideas. It is a special case of a more general two-phase process, called Abstract Programming, in which an informal and imprecise specification is transformed into a formal abstract operational program, which is then transformed into a concrete (i.e., detailed low-level) program by optimization. Abstract programming thus consists of a specification phase and an implementation (optimization) phase which share a formal abstract operational program as their common interface.

The concept of abstract programming is completed by adding the feedback loops required by testing, maintenance, and tuning. In conventional programming, where no abstract program exists, these feedback loops all operate on the optimized concrete program. On the other hand, in abstract programming, if an effective method can be found for guaranteeing the validity of an implementation (that is, the functional equivalence of the abstract and concrete programs), then the validation process can be shifted to the specification phase to show equivalence between the user requirements and the abstract program. Thus, validation could, and should, occur before any implementation. Furthermore, if the implementation process could be made inexpensive through computer aids, then maintenance could be performed by modifying the specification and reimplementing it rather than directly modifying the optimized concrete program, as is current practice. The importance of such an advance can be recognized when one realizes that optimization is the process of maximally spreading information (to remove redundant processing), and that modification requires information localization. Thus, the two processes are diametrically opposed; this fact explains much of the current problem with modifying and maintaining existing programs. The second major cause of this dilemma

is that optimization obscures clarity and thus makes it difficult for maintainers even to understand how the concrete program operates.

It is therefore clear that major advances in programming will hinge on the ability to provide an inexpensive optimization process with guaranteed validity so that maintenance and validation can occur in the specification phase on the abstract program (as shown in Figure 3.1) rather than in the implementation phase on the concrete program.

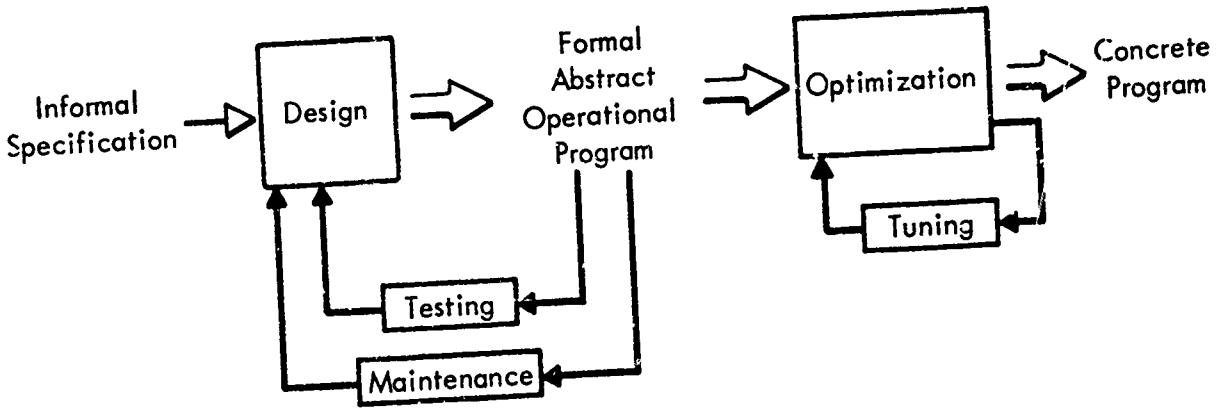


Figure 3.1 Abstract programming

The key element of the whole abstract programming approach is the abstract operational program itself. Currently, considerable effort is directed toward designing appropriate languages for writing such abstract programs; however, no matter how "high-level" these languages become, they are formal programming languages, and as such demand unambiguous, complete, and consistent specifications. It is just these demands that make programming difficult and that make necessary a tool to aid specifiers in buiding suitable formal specifications.

Currently, all large software systems are specified in written form at a number of levels of detail before implementation begins. One such level is the "functional specification," which describes in natural language how the system is supposed to operate and represents a first-level design. These specifications are currently used by programmers as the description of what to implement. Unfortunately, problems arise because the specifications, although generally understandable, are generally neither complete, unambiguous, nor consistent. The programmer is left to his own ingenuity to discover these problems and to fix them himself or to obtain a clarification from the specifier.

PROJECT GOALS

The Specification Acquisition project has therefore adopted as its goal the development of a system that aids system specifiers in converting their informal specifications into a precise operational abstract program. To produce such a system we needed both an understanding of the structure and content of such informal specifications and a theory of how they could be formalized. We therefore first undertook an extensive survey of existing military functional specifications (as embodied in Military Standard-490 B5 specifications). These natural language descriptions represent a first-level design of the intended system. They apportion the required processing into modules and describe the interfaces between the modules and the overall control structure. Because the audience for these descriptions is other people (as opposed to computers), they employ the full variety of detail suppression mechanisms found in natural language, including omitted parameters to actions, anaphoric reference, implicit or omitted control structure, terminology shifts, part/whole interchangeability, etc. (These mechanisms are more fully described in the Appendix.) The reader of the specification is required to amplify the text and determine for himself which details have been suppressed.

Our study of these specifications and their detail suppression mechanisms led to our proposed theory of how people understand such specifications and fill in the suppressed details. Our theory is simply that these specifications are understood not in a general natural language context, but rather in the much more specific context that an operational program is being specified. Understanding these specifications basically requires that the correct interpretation of each statement is chosen from several possible interpretations of the natural language statement. The key to our theory is that these choices are guided by the statement's use in the operational program. Fortunately, programs are highly constrained objects (one reason it is so hard to construct them) and therefore act as effective filters of possible interpretations. In Artificial Intelligence terms, program understanding is a domain of strong semantic support.

The SAFE system is based on this theory. It forms the individual statements into a program schema and then attempts to "run" the schema. Symbolic rather than actual data is used as input so that general program behavior can be analyzed. At each step in this "running" of the program a particular interpretation must be chosen for the current statement before it can be executed. The chosen interpretation is accepted if and only if it does not cause any violations of the rules of well-formedness of programs, which are of three forms. First, the program must pass a set of static (syntactic) well-formedness rules such as "parameters must be used in a routine" and "the type of an actual argument must agree with the corresponding formal parameter type." Second, the program must pass a set of dynamic (semantic) well-formedness rules, such as (1) "if X is performed for the purpose of Y then Y must use the results of X," or (2) "the predicate of an IF statement cannot be determinable from the program itself" (if it were, then its evaluation is

independent of the actual input and therefore not really a conditional as expected). Finally, the program behavior cannot violate any constraints of the domain.

Whenever one of these rules is violated, the "run" is backed up to the last choice point and a different choice is attempted (if no possibilities remain, then the previous choice point is tried). This process is continued until either a successful "run" is obtained or all possibilities have been exhausted. Interpretations are thus chosen and evaluated in the context of how they are used in the run-time environment of the program. The process of "running" a program with symbolic inputs, called meta-evaluation, has been extensively developed by researchers interested in proving properties of programs. As far as we know, this project represents the first use of meta-evaluation for program understanding rather than program proof.

PROGRESS AND ACCOMPLISHMENTS

A major milestone was achieved when the SAFE system transformed the informal functional specification shown in Figure 3.2 into an operational program. The informality of this specification is shown in Figure 3.3, which indicates some of the suppressed details, terminology conflicts, and ambiguous constructs contained in this example. The formal program produced as the precise specification of the input in Figure 3.2 is shown (in a simplified publication syntax) in Figure 3.4.

This example was extracted from an actual Army functional specifications manual. The original specification was much larger and more complex, but the simplified version retains the essential character and style of the original. It should be noted from Figure 3.2 that the actual input is parenthesized (each noun-phrase and verb-phrase is parenthesized) so that syntactic parsing considerations can be avoided. Such parenthesization does not, however, solve or mitigate the semantic interpretation issue discussed above (nor those shown in Figure 3.3) which still remain for the system to resolve by meta-evaluation.

In an effort to determine how difficult it is to understand this particular informal specification and convert it to an operational abstract program, we asked several staff members to construct an abstract program corresponding to this specification. We asked them not to optimize the abstract program because our system is not concerned with efficiency issues.

The problem was much more difficult than we suspected. The subjects averaged eight hours to accomplish the task and each subject had either design or coding errors, or both. Furthermore, the programs produced were approximately the same size as that produced by the system. One interesting result of this test is that the system made one

error--also made by one of the subjects. The error was that copies of the edited message were distributed rather than copies of the original; it was caused by the fact that the system didn't understand that distinctions between original and current state are frequently suppressed, and because its current analysis of the usage of produced results is very simple. Both of these deficiencies are expected to be corrected in the next version.

PLANS

Though the results using this example are very promising, and although we have attempted to build a general system capable of handling a wide variety of specifications from many different domains, it is extremely difficult to extrapolate from a single data point. We therefore are planning to present several different and more complex examples to the system during the next year.

```
* ((MESSAGES ((RECEIVED) FROM (THE "AUTODIN-ASC"))) (ARE PROCESSED) FOR (AUTOMATIC DISTRIBUTION
ASSIGNMENT))

* ((THE MESSAGE) (IS DISTRIBUTED) TO (EACH ((ASSIGNED)) OFFICE))

* ((THE NUMBER OF (COPIES OF (A MESSAGE) ((DISTRIBUTED) TO (AN OFFICE)))) (IS) (A FUNCTION OF (WHETHER
((THE OFFICE) (IS ASSIGNED) FOR (("ACTION") OR ("INFORMATION"))))))

* ((THE RULES FOR ((EDITING) (MESSAGES))) (ARE) (: ((REPLACE) (ALL LINE-FEEDS) WITH (SPACES)) ((SAVE)
(ONLY (ALPHANUMERIC CHARACTERS) AND (SPACES))) ((ELIMINATE) (ALL REDUNDANT SPACES))))

* (((TO EDIT) (THE TEXT PORTION OF (THE MESSAGE))) (IS) (NECESSARY))

* (THEN (THE MESSAGE) (IS SEARCHED) FOR (ALL KEYS))

* (WHEN ((A KEY) (IS LOCATED) IN (A MESSAGE)) ((PERFORM) (THE ACTION ((ASSOCIATED) WITH (THAT TYPE OF
(KEY))))))

* ((THE ACTION FOR (TYPE-0 KEYS)) (IS) (: (IF ((NO OFFICE) (HAS BEEN ASSIGNED) TO (THE MESSAGE) FOR
("ACTION")) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS ASSIGNED) TO (THE MESSAGE) FOR ("ACTION"))) (IF
((THERE IS) ALREADY (AN "ACTION" OFFICE FOR (THE MESSAGE))) ((THE "ACTION" OFFICE FROM (THE KEY)) (IS
TREATED) AS (AN "INFORMATION" OFFICE))) (((LABEL OFFS1 (ALL "INFORMATION" OFFICES FROM (THE KEY))) (ARE
ASSIGNED) TO (THE MESSAGE)) IF ((REF OFFS1 THEY) (HAVE (NOT) (ALREADY) BEEN ASSIGNED) FOR (("ACTION") OR
("INFORMATION"))))))

* ((THE ACTION FOR (TYPE-1 KEYS)) (IS) (: (IF ((THE KEY) (IS) (THE FIRST TYPE-1 KEY ((FOUND) IN (THE
MESSAGE)))) THEN ((THE KEY) (IS USED) TO ((DETERMINE) (THE "ACTION" OFFICE)))) (OTHERWISE (THE KEY) (IS
USED) TO ((DETERMINE) (ONLY "INFORMATION" OFFICES)))))
```

Figure 3.2 Actual input for message processing example

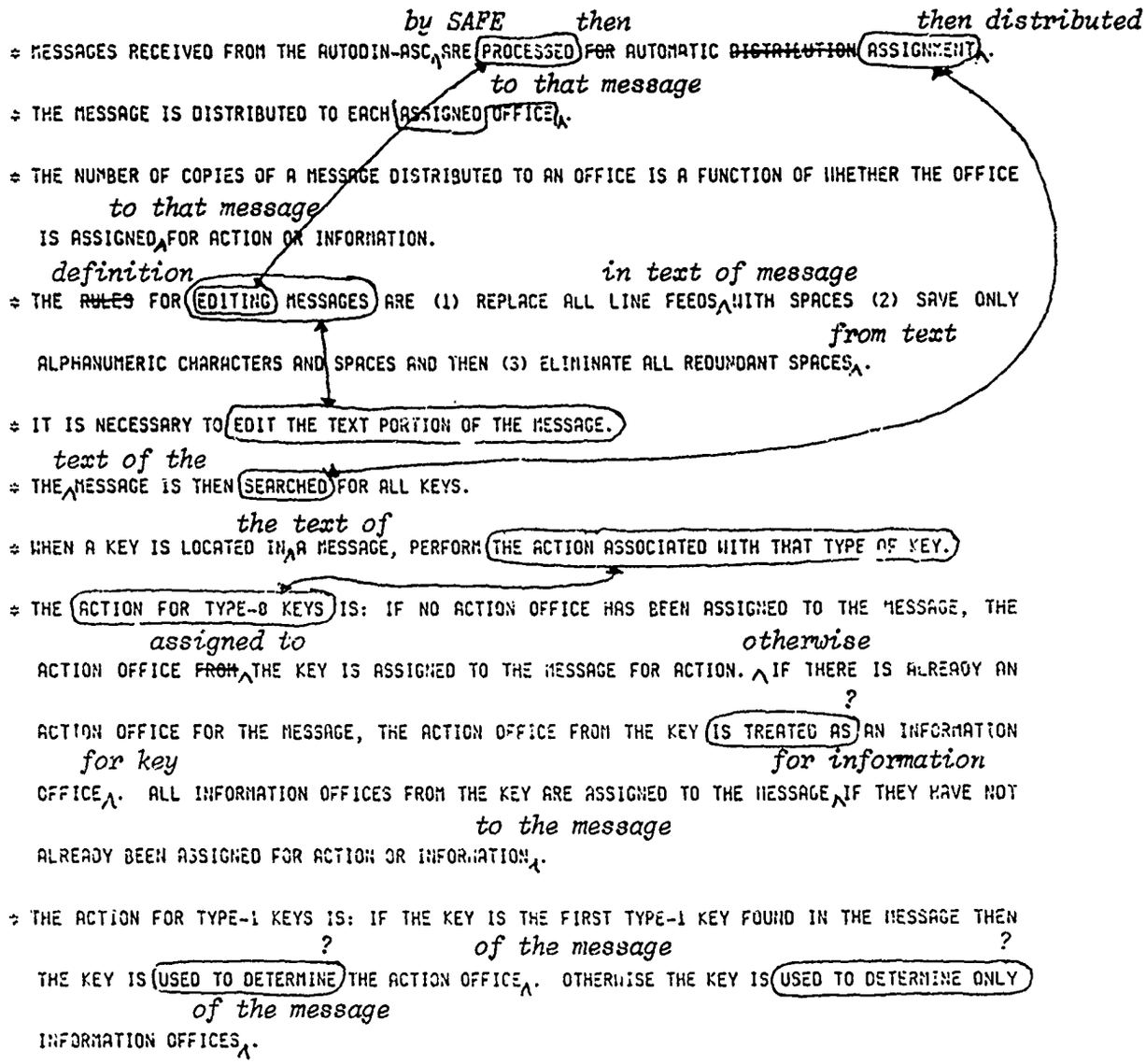


Figure 3.3 Specification deficiencies of message processing example (by conventional programming standards)

```
(WHENEVER (receive message FROM autodin-asc BY safe)
          (edit text OF message)
          (search text OF message FOR (CREATE THE SET OF keys))
          (distribute-process#1 message))
(distribute-process#1 (message)
      (FOR ALL offices WHICH ARE (assigned office TO message FOR ANYTHING)
           (distribute-process#2 message office)))
(distribute-process#2 (message office)
          (FOR (function#1 (TRUTH-VALUE OF (assigned office TO message FOR action))
                           (TRUTH-VALUE OF (assigned office TO message FOR information)))
          TIMES (distribute A copy WHICH IS A copy OF message AND located AT safe
                            FROM safe TO location OF office)))
(edit (text)
    (FOR ALL line-feeds WHICH ARE IN text
       (replace line-feed IN text BY (CREATE AN ORDERED SET OF spaces)))
   (keep THE (union (CREATE THE SET OF alphanumeric characters IN text)
                    (CREATE THE SET OF spaces IN text))
             FROM text)
   (FOR ALL spaces WHICH ARE IN text AND redundant IN text
            (remove space FROM text')
(WHENEVER (locate P key IN text OF message AT POSITION ANYTHING)
          (CASE (type OF key)
                (type-0 (type-0-action message key))
                (type-1 (type-1-action message key))))
(type-0-action (message key)
     (IF (NOT (EXISTS action office FOR message))
       THEN (assign THE action office#1 FOR key TO message FOR action)
       ELSE
          (treat action office#2 FOR key AS information office#2 FOR key
          IN (IF (NOT (assigned office#2 TO message FOR action OR information))
              THEN (assign office#2 TO message FOR information))))
      (FOR ALL office#3 WHICH ARE (assigned office#3 TO key FOR information))
          (IF (NOT (assigned office#3 TO message FOR action OR information)
            THEN (assign office#3 TO message FOR information))))
(type-1-action (message key)
    (IF key = (key#1 WHICH IS (SEARCH HISTORY FOR FIRST
                (locate type#1 key#1 IN text OF message AT position ANYTHING)))
        THEN (determine THE action office FOR message
                BY (type-0-action message key))
        ELSE (determine ONLY THE information office FOR message
             BY (IF (EXISTS action office FOR message)
                THEN (treat action office#1 FOR key AS information office#1 FOR key
                  IN (IF (NOT (assigned office#1 TO message FOR action OR information))
                    THEN (assign office#1 TO message FOR information))))
                (FOR ALL office#2 WHICH ARE (assigned office#2 TO key FOR information)
                     (IF (NOT (assigned office#2 TO message FOR action OR information))
                         THEN (assign office#2 TO message FOR information))))))
```

Figure 3.4 Program created by prototype system

APPENDIX

WHAT NATURAL LANGUAGE CONSTRUCTS ARE IMPORTANT FOR PROGRAM DESCRIPTIONS

Natural language program descriptions seem to depend upon a particular model of the (relevant) world populated by objects of various types having certain attributes which are relationships between themselves and other objects. This collection of objects and their relationships can be classified as a model because the objects must obey certain rules (both static and dynamic) that limit the allowed states the model can adopt. These rules are the "physics" of the model.

With such a model as a basis, a program description corresponds to rules for forming sequences of actions that will guarantee that the model's behavior (as constrained by the physics) additionally meets some other criteria--the goal of the process. The behavior is specified either as the final state of the model or as the succession of states it assumes. The sequence formation rules are the analog of control statements in conventional programming languages.

Thus, as a first approximation, a language suitable for natural language program descriptions must be capable of defining types of objects, instances of these objects with specific attributes and relationships with other instances, constraints on the allowable states that the collection of objects can assume, actions which modify the objects and their interrelationships, and rules for forming sequences of these actions to achieve some goal. Such a language closely approximates the current notion of an abstract programming language.

The natural language model differs from the abstract program model in two fundamental ways: First, in the natural language model, it is assumed that any desired information concerning the current or previous states of the model can be directly retrieved from the model; thus all information is viewed as primary. Second, all the consequences of an action are not specified as part of the action's description, but only the portion directly related to the actions. The others can be derived from those specified. These rules of information derivation are called inference rules, and they permit the natural language model to ignore the distinction between primary and derived information and to specify only the primary effects of an action. They also permit the derivation of information to be decoupled from both how the information used in the derivation was itself generated and how the desired information is used. They thus deal with the consistency of the model in any state rather than the movement between states. The task of maintaining the consistency of the state is transferred to the system through these inference rules. Thus the natural language model must embody inference rules and

must use them to maintain the consistency of the model state as it is changed by actions. Using this natural language model as a basis, we can now discuss the three broad categories--declarative, naming, control--of constructs which appear in natural language program descriptions.

Declarative Information

Declarative information is used to help the translator select the appropriate interpretation of procedural constructs. Most of this information (such as type, relation, constraints, and inference rules) has already been described above. The issues here are that in natural language descriptions this information is mixed with the procedural description rather than separated from it and the use of the information extends beyond static characteristics to dynamic behavioral ones. Thus, constraints are often used to help eliminate interpretations that will cause the constraint to be violated.

There is one additional type of declarative information, called expectations, not already discussed. These are promises that certain things will occur or that the program being described fits together in certain ways. For example [all the examples are drawn from the program description in Figure 3.2]:

- "The message is processed for distribution" - This sets up the expectations that processing precedes distribution and that the processing produces some results used by distribution.

As with the other declarative information, these expectations can be used to eliminate possible interpretations which do not fulfill the expectations.

Naming Constructs

In programming languages variables are normally given unique names or--if names are reused--precise scoping rules provide a unique interpretation. In natural language, on the other hand, objects are almost never given names; instead, context in conjunction with a naming construct provides a unique reference. Thus both a context mechanism and the following variety of naming constructs must be provided.

A. ***Descriptive.*** A set of associations is given which uniquely selects a particular object from those in the current context or the set of all objects of the specified type. For example:

- "The office assigned to the key for action" - If it is assumed that the reference to "key" is unique and that only one office is assigned to that key for action, then this descriptive reference is also unique.

B. ***Simple typed references.*** Only the type of the desired object is specified; the resolution must be provided either by a unique instance of that type in the context or by the operations performed on the object. For example:

- "The message is distributed" - Context is used to determine that the received message is the desired one.

- "Replace all line-feeds" - Using the line-feeds (i.e., replacing them by spaces in the text of the message) requires that the referenced line-feeds be in the text. Thus only line-feeds in the text satisfy this typed reference.

C. ***Approximate references.*** The stated reference doesn't specify the necessary relations to determine the desired object, which can be either explicit as in the generalized prepositions below or hidden so that only a failure of the referenced object to satisfy some usage criteria indicates the necessity to reinterpret the reference as an appropriate reference to some closely related object. Resolution of this type of reference requires both context and usage analysis. Two broad subcategories are:

C1. ***Generalized prepositions.*** When a strong association exists between objects, a preposition may sometimes be used in place of the association to relate the objects.

- "The action office from the key" - The word "from" indicates that some unspecified relation associates office, action, and key. In this example, this association is the office assigned to the key for action. The reference is then treated as descriptive.

C2. ***Hidden.*** Reference is made to an object when a closely related one (in this case a subpart) is desired.

- "The message is then searched" - Although "message" is specified, only the text portion has the appropriate attributes to be searched; thus it is the intended reference.

D. ***Missing operands.*** In many situations operands to associations or actions are completely omitted from the natural language program descriptions. Such

omissions are possible only when the context and usage constraints are strong enough to determine the reference, or when the particular reference is irrelevant or the entire class of possible referents is intended. In addition to using such context and usage information, the omission must be recognized to initiate these mechanisms; such recognition depends upon knowing what operands are required for each association and/or action used.

- "The message is distributed to each assigned office" - The "assign" association requires three operands: an office, a type of assignment (either action or information), and an object being assigned to (either message or key). Only one of the three--office--is specified. Thus the object being assigned to is omitted and context is used to determine that the message should be distributed only to those offices assigned to that message. Similarly, context is used to determine that offices assigned for either action or information (the only two possibilities) are intended for distribution.

Control

As with naming, natural language program descriptions use several control constructs which do not exist in programming languages and which permit a very different organization for these descriptions. Instead of a whole composed of highly structured and tightly connected parts, as in programming languages, they form a loose confederation of unconnected fragments held together only by constructive efforts based on context and usage.

This constructive effort is exactly that needed to transform the description into the program control structure. It should not be surprising that the control structure has been suppressed and dispersed in descriptions because these descriptions are intended primarily for understanding, not execution. Program descriptions thus highlight the processing for the normal case by presenting it first, followed by descriptions of any exceptions and how these exceptions should be processed. They in turn are similarly described, so that any exceptions to an exception will follow the description of that exception.

The structure, then, of natural language program descriptions is that a fragment (a piece of the description with internal conventional control structure) is followed (not necessarily immediately) by a set of exceptions that specify the application criteria and the processing (as a fragment) for those cases which satisfy the criteria. This is exactly opposite to the order found in programming languages, in which all the special cases precede the normal case. Details are similarly suppressed in a fragment to promote

understanding, and the amplifications occur later in fragments which indicate what is being amplified and what the amplification is.

Constructing the program control structure for a program description thus involves integrating the fragments, which requires that the type of each fragment be identified as a normal case, exception, or amplification specification. For each fragment of the latter two types, the fragment being modified or amplified must be identified (this is a naming problem similar to those described above).

Amplification can be treated as substitutions for the named construct within the amplified fragment. There are, however, two problems with exceptions. The first is the relative ordering of multiple exceptions to a fragment. Normally such ordering is not explicitly stated and must be determined by the type of overlap between their applicable cases and/or processing assumed in the modification. As part of this ordering problem, it must also be determined (by a similar analysis) whether or not a case that satisfies the criteria for an exception should then be tested against the criteria of other exceptions and/or processed by the normal case. The second problem with exceptions is that the processing to be performed might not be directly specified, but rather given as a modification of or a similarity to already specified processing, which requires that a new category of "editing" operations (such as bypass, include, except, treat as, inhibit, enable, etc.) must be understood and handled.

4.
PROTECTION ANALYSIS

Research Staff:
Richard Bisbey II
Jim Carlstedt
Dennis Hollingworth
Dale Chase

Consultant:
Gerald J. Popek

Support Staff:
Nancy Dechter

INTRODUCTION

The Protection Analysis Project is an ongoing research effort toward improving the security of existing general-purpose resource-sharing operating systems by finding errors in their protection mechanisms. This task has come to be called "protection evaluation." The problem is of obvious importance in view of the investment existing systems represent, their expected lifetime, and their insecurity. It is well known that current general-purpose operating systems, due to their size and complexity, usually contain a large number and variety of errors even after having been in service for years. These include security errors--indicated by the fact that skillful penetration efforts directed against these systems invariably succeed. The task of improving such systems is urgent, since many are installed in governmental, commercial, and military environments in which the requirement for security (in terms of the magnitude of losses from accidental or intentional violations) is strong and immediate. These losses will be reduced in proportion to the cost-effectiveness of the available error-finding tools.

APPROACH

There are several possible approaches to the elimination of protection errors in general-purpose operating systems. Probably the most elegant and formal is the use of program verification techniques, i.e., proving a program correct with respect to a collection of mathematical assertions. To use program verification for protection evaluation, three tasks must first be completed:

1. The protection policy to be enforced must be formalized.

2. The formalized protection policy must be proved complete and consistent.

3. The formalized protection policy must be translated into program verification assertions.

Program verification techniques would then be used to prove the assertions correct with respect to the operating system code.

There are, however, numerous problems associated with the above approach. First, no one has been able to write the formal protection policy for an operating system, let

alone address the problems of proving that policy consistent and complete and of translating it into program verification assertions. Furthermore, current program verification techniques are not powerful enough to handle the language constructs used in contemporary general-purpose operating systems. Finally, there is the related problem that contemporary operating systems are too large and complex to be considered as viable candidates for verification.

Research is being directed to the solution of each of the above problems. For example, research is currently under way in formulating protection policy. Examples include the "star-property" [Bell 73] and "data security" [Popek 76]. Considerable effort is also being expended to improve and automate program verification techniques* and to develop programming languages for producing operating systems which are more amenable to program verification techniques [Lampson 76]. Finally, research is being directed at decreasing the size of existing systems [Schroeder 75] and creating systems based on small kernels [Panel 74].

While an approach based on the use of formal techniques is desirable, the aforementioned problems prevent their application to contemporary systems. What then can be done? Two basic problem areas require attention:

1. Derivation of the policy against which the correctness of an operating system's protection mechanisms will be judged.

2. Development of less formal evaluation procedures to compare the protection policy with more immediately applicable operating system code.

The Protection Analysis project is addressing these problems in the following ways.

Derivation of Enforcement Policies

While it is not presently possible to specify a total protection policy, let alone verify its completeness and consistency, there is a basis for empirically identifying elements of that policy from observed protection errors in existing systems. Because protection errors are, in fact, nothing more than violations of some protection policy, from a collection of observed protection errors it is possible to derive the protection policies violated.

Identification of Evaluation Tools and Techniques

The second problem is to evaluate the protection mechanisms of operating systems with regard to the derived policies. As stated previously, general-purpose program verification techniques are not powerful enough to handle programs of the size and complexity of contemporary general-purpose operating systems. However, it is possible to develop for a single, specific element of protection policy a special-purpose evaluation tool or technique.

---

*See Section 1 of this report.

In summary, the ISI approach consists of deriving protection policies from known protection errors and developing individual search techniques capitalizing on these policies. (For a more detailed discussion of the issues involved, see [Carlstedt 75].)

PROGRESS

During the past year, an analysis was completed of a collection of known protection errors in operating systems. The goal of this analysis was to categorize the errors as to type. As a result of the study, it was found that all of these errors fell into one or more of ten categories.

The remaining research effort has focused on the generation of operating system evaluation procedures. The three error categories for which investigations have been completed are: consistency of data over time; validation of operands; and residuals. Research continues for other categories.

Consistency of Data over Time

Operating systems continuously make protection-related decisions based on data values contained within the system data base as well as on values which have been submitted to and validated by the system. In order for a correct protection decision to be made (in the absence of other types of protection errors), the data must be in a consistent state, i.e., the value or structure of the data on which the protection decision is made must be in some specific relationship with other data in the system, and must remain in that relationship during the interval in which the protection decision is made.

Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time [Bisbey 75] describes the general error type as well as a particular instantiation, i.e., those errors resulting from inconsistencies in parameters supplied to the operating system.

Validation of Operands

Within an operating system, there are numerous operators responsible for maintaining the system's data base and for changing the protection state of processes or objects known to the system. Many of these operators are critical in the sense that if invalid or unconstrained data are presented to them, a protection error results.

Protection Errors in Operating Systems: Validation of Critical Conditions [Carlstedt 76] investigates the validation of conditions attached to critical operators and their operands, especially when those operands can be readily influenced by users. A companion research report [Bisbey 76] describes a specific technique, Data Dependency Analysis, for automatically finding data flow paths within operating systems, thereby making it easier to detect errors of this type.

Residuals

A widely recognized error type is that of the "residual," i.e., information which is "left over" in an object when the object is deallocated from one process and allocated to another. Several types of residual errors exist, including

- *Access Residuals.* Incomplete revocation or deallocation of the access capabilities to the object or cell.

- *Attribute Residuals.* Incomplete destruction of the cell's context with other cells or objects, and of old values within the cell.

Protection Errors in Operating Systems: Allocation/Deallocation Residuals [Hollingworth 76] examines the sources of these errors and discusses strategies for them in operating systems.

Future research effort will focus on the development of evaluation procedures for finding other major error types.

IMPACT

The work described here in will have an impact in several areas, most immediately in the evaluation of existing operating systems with respect to the reliability of their security mechanisms. The empirical basis of the research makes it easy to incorporate new error types and detection techniques should they be identified. The evaluation techniques can also be used in computer acquisition as part of a set of standard tests for system acceptance. Additionally, the data base of errors and error types is useful in the repair or modification of existing systems; it could form the basis for a "best practices manual," i.e., a discussion of errors and problems that should be avoided in the design of future systems and protection mechanisms. Finally, the analysis needed to derive error types and to develop associated error detection algorithms yields insights that contribute to a deeper understanding of protection itself.

REFERENCES

[Bell 73] Bell, D., and L. LaPadula, *Secure Computer Systems: A Mathematical Model*, ESD-TR-73-278, Vol. II, November 1973.

[Bisbey 75] Bisbey II, R., G. Popek, and J. Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time*, USC Information Sciences Institute, ISI/SR-75-4, December 1975.

[Bisbey 76] Bisbey II, R., J. Carlstedt, D. Chase, and D. Hollingworth, *Data Dependency Analysis*, USC Information Sciences Institute, ISI/RR-76-45, February 1976.

[Carlstedt 75] Carlstedt, J., R. Bisbey II, and G. Popek, *Pattern-Directed Protection Evaluation*, USC Information Sciences Institute, ISI/RR-75-31, June 1975.

[Carlstedt 76] Carlstedt, J., *Protection Errors in Operating Systems: Validation of Critical Conditions*, USC Information Sciences Institute, ISI/SR-76-5, May 1976.

[Hollingworth 76] Hollingworth, D., R. Bisbey II, J. Carlstedt, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals*, USC Information Sciences Institute, ISI/SR-76-7, June 1976.

[Lampson 76] Lampson, B., J. Horning, R London, J. Mitchell, and G. Popek, *Euclid Report*, Xerox Palo Alto Research Center, April 17, 1976, Draft.

[Panel 74] Panel Session--Security Kernels, AFIPS Conference Proceedings, National Computer Conference, Vol. 43, AFIPS Press, 1974, pp. 145-151.

[Popek 76] Popek, G., and D. Farber, *A Practical Example of Program Verification*, University of California at Los Angeles, 1976, submitted for publication.

[Schroeder 75] Schroeder, M., "Engineering a Security Kernel for Multics," Proceedings of the Fifth Symposium on Operating System Principles, *ACM Operating Systems Review*, Vol 9, No. 5, November 1975.

5.
INFORMATION AUTOMATION

Research Staff:
Donald R. Oestreicher
Robert H. Stotz
John Collins
John F. Heafner
Robert T. Martin
Jeff Rothenberg
Ron Tugender
Dono van-Mierop
John J. Vittal

Research Assistant:
Larry Miller

Support Staff:
Katie Patterson

INTRODUCTION

The increasing sophistication of weapons systems and decreasing time frame for making decisions make it essential to provide the military commander better quality information faster, even though manpower has been reduced by the conversion to all-volunteer forces. With today's technology, messages can traverse several thousand miles in fractions of a second, but hours are lost at either end, both in entering the message into the communications system and in delivering it to the man who can act on it.

The IA project is studying the application of on-line, interactive computer technology to the military message handling problem and is preparing an operational test for a system designed to help solve the problem. On the basis of the ARPANET message system experience, we are confident that such a service has a high payoff to the military. Not only can formal message preparation and delivery become faster and more reliable, but the processing facilities provided can also be put to new use; for example, with such a service the status of a message is automatically available at all stages from preparation to delivery. Much more detailed accounting and auditing is easy to maintain, providing a better understanding of the basic communication process. Entirely new facilities become available as well: for example, using the message service to alert individual users when certain events have occurred (e.g., "the message from Capt. Jones that you were expecting has arrived"). Automated suspense files, calendars, etc. are also simple to provide.

Perhaps the most important contribution of such a system is that it makes available a secure, informal (off-the-record) message facility. This "electronic memo pad" is swift and convenient to use and, unlike the telephone, does not require the simultaneous attention of sender and receiver.

The project is specifically directed to the military communication environment, and even more specifically to nonexpert users. The most effective way to introduce such a

service into the military community is by means of an operational test at a military site, which will serve a twofold purpose: It will demonstrate the utility of an on-line message service in an environment credible and comprehensible to military planners, and allow system planners to understand the impact of such a system on the user organization and to evaluate the cost versus benefits of its various features. System builders will obtain a better understanding of the implementation and delivery issues.

BACKGROUND

Although the IA project actually began in the fall of 1973, its roots reach back to a five-week study, conducted on behalf of ARPA, of the military communications on the island of Oahu [1]. This study was initiated at the request of the Secretary of Defense for Telecommunications as a part of a Navy program called COTCO, whose mission was to consolidate and improve communications on Oahu. Until ARPA'S involvement, COTCO advocated conventional data processing solutions. The ISI report recommended a complete island-wide interactive writer-to-reader message service electrically coupled to AUTODIN (the military's backbone communication system); it was submitted by ARPA to DoD, where it excited considerable interest but was generally regarded as too radical to be included in a production system without a better appreciation of its cost and benefits.

Recognizing this, ISI cooperated with ARPA in developing a sensible military message program and in making the case to the military establishment for the usefulness of a test of an on-line interactive message service in an operational military environment. From such a test one can learn what features are valuable, how the service is used, and how it affects the way the user organization does its business. This information is essential for long-range military communication planning and for proper implementation of production systems.

By the summer of 1975 ARPA organized a separate program for military message handling. In December 1975 the effort culminated with the signing of a memorandum of agreement between Commander-in-Chief, Pacific (CINCPAC); Commander, Naval Electronics Systems Command (NAVELEX); Commander, Naval Telecommunications Command, (NAVTELCOMM); and Director, Defense Advanced Research Projects Agency (DARPA). This memorandum calls for a test of such an experimental message service to be run at CINCPAC staff Headquarters, Camp Smith, Oahu The experiment is designated to begin in January 1977 and to last for two years. Funding is to be shared jointly by NAVELEX and DARPA.

INFORMATION AUTOMATION PROJECT

The IA project was started at ISI in the fall of 1973 with a twofold goal: 1) to develop the technology for providing on-line computer services directly to users who are

neither specialists in computer science nor specifically trained operators and 2) to develop an on-line, interactive, writer-to-reader message service for the military community. The two goals are in fact indivisible. The military action officers who send and receive messages are not computer specialists. For the service to be useful, an interface must be provided that knows a great deal about each individual's habits, thus making his use of the service seem easy and natural to him.

The military have been users of an electronic message service for many years. At each organization the service has always been handled as an over-the-counter business, an outgrowth of its development from telegraphy. Over this long history much procedure and policy has developed. It is the over-the-counter, manual handling of messages that the on-line service is designed to replace. This new system will bring a new style of operation which will affect some of the old policy and alter much of the procedure. To be a success, an on-line message service must provide the improvements inherent in automation without overly disrupting the traditional patterns and procedures that are known to work. The manual nature of today's message service is somewhat cumbersome, but it is extremely flexible; each command or organization is able to tailor its procedures to its own needs. One of the unique goals of the IA message service is to provide this tailorability.

To adequately support military message handling the organizational structure of the user community must be reflected in this service. For example, the rules about who can access what message files and who can release what messages must be carefully modelled. By definition, formal military message traffic flows between commanders of organizations, even though the messages nearly always originate and terminate at much lower levels. This necessitates special "coordination" or "staffing" procedures on outgoing messages (which require approval up the entire chain of command) and complicates the distribution of incoming messages. The IA military message service is unique in its approach to these problems, making it possible to retain these formalisms to any degree found necessary in the tests.

It is also necessary that the on-line service be easy to use. It is certainly easier to type "send for coordination" than to hand-carry a draft message around to each coordinator. However, by automating this transmission we are faced with making the use of terminals competitive with paper and pencil. Toward this end the IA project is developing scanning and editing aids that currently do not exist. For instance, to facilitate integration of comments and changes from several coordinators, the service offers the ability to compare two versions of the same paragraph on separate windows of the CRT screen, highlighting the differences by making the changed characters brighter.

The proposed IA message service is divided into two stages: preparation and delivery. The former stage includes the creation of the draft message and the coordination of this draft with other users until it is signed off for release. For this stage

the IA message service provides a special-purpose editing program which understands message formats and checks that the contents of the various fields are legitimate. The editor is structured so that a coordinator's editing of a message is stored as a special change file rather than as actual modifications to the original.

The author of the draft message controls the sequence and timing of delivery of the draft to coordinators. The message can proceed serially or in parallel (or any combination of the two). The author can have the message returned to him after each signoff (so he can incorporate the changes), he can ask that he simply be notified after each signoff, or he can let the coordination delivery proceed automatically.

Often a coordinator of a message wishes to obtain the opinions of others on his staff before he signs off. The IA message service allows the coordinator to "delegate" to as many people as he wishes the capability to comment and edit the message (each delegate edits from the original and creates his own change file). If so inclined, the coordinator may also delegate the signoff responsibility, but this is restricted to a single delegate only. The message service may retain all of this delegation information for audit purposes. It is planned that this delegation facility will be extended to permit a user to specify in advance the criteria for selecting messages to be delegated to others. The service will then automatically perform the delegation whenever a message meeting these criteria is received.

This coordination process can be iterated as often as necessary, with each version being coordinated independently. A major IA research goal is to learn more about the staffing process and about how to structure the computer-aided environment to enhance the effectiveness of this coordination.

The delivery stage involves conveying the message to its ultimate recipients, archiving it, plus providing aids for the user to sort his messages, scan them, and file them for later retrieval. The first step in this process is to determine distribution for the message. Because of the military policy that all formal traffic flows between commanders of organizations, it is necessary to employ complex procedures to determine the real ultimate recipients. The IA message service extends the normal "one-pass" distribution algorithms provided in current AUTODIN terminals (e.g., LDMX) to allow each user to build his own automatic personal distribution determination. A special form of distribution determination provided by IA, called Guarding, allows a user to specify criteria for messages that are to be routed to the first "on-line" user on the guard list. This assures that incoming messages meeting these criteria will be delivered to a live person who can act on it immediately.

In addition to determining the distribution of a message, it is military policy to assign one of the recipients responsibility for taking whatever action is appropriate. The officer assigned the "Action" may further delegate the Action to a subordinate or may "sell the action" to some other more appropriate officer. The automated message service must keep track of this action assignment until such time as the action has been completed.

A different form of special handling offered by IA is the alerting mechanism, which allows users to specify criteria for messages that will cause immediate action on the user's screen when they are received. This will notify the user of the event immediately, if he is on-line, or as soon as he comes on, if the event occurred while he was off-line. A further advantage is that wherever he is he may get on-line via any available terminal.

Message selector criteria can also be applied to incoming messages to sort them into "folders" for the user, which provides the electronic analog of file cabinets. Since the message service can retrieve messages rapidly, these users' folders actually store only citations to messages rather than the messages themselves, which reduces the computer storage required to easily manageable size. The IA Message Service provides tools for specifying "key words" which can be used for later retrieval of the message.

User Support

The ARPANET experience provides ample evidence that computer scientists can use on-line systems effectively with little or no formal training in their operation. There are also many examples of systems used every day by nonspecialists who have had intensive training (e.g., airline reservation clerks). To be effective, however, a military message service must be usable by non-computer people (action officers) with minimal formal training. Few officers spend more than 10 percent of their time in message-related functions; moreover, the present effort requires no specialized training. No on-line message service will be used in the military if it is not virtually self-evident and highly supportive whenever the user has any questions or difficulty. The IA project is focusing on this problem as a central research issue.

The approach chosen to provide the necessary support for the user who is not a trained operator or a computer specialist is to interface him to the message service through an "intelligent front-end process" which we call his "Agent." This Agent makes the service appear consistent to the user. It is designed to handle all control procedures (e.g., editing, help, defaulting, error handling, context mechanisms, etc.) in the same place and therefore in the same way throughout all phases of the service. The lack of such consistency is a major source of difficulty in the current TENEX message facilities. The Agent and its components are described in detail in [2, 3, 4]. Briefly, it consists of a Command Language Processor, a User Monitor (with attendant background analysis processes), and a Tutor.

Command Language Processor (CLP). This serves as the interpreter for user commands operating from a dynamic input string and provides input editing functions and screen control. To support the neophyte, the CLP has a strong emphasis on error detection, recovery, and correction. It also acts as the driver for the rest of the Agent, calling in the User Monitor and the Tutor when appropriate. The CLP operation is affected by User Profile data which provides information unique to each user.

User Monitor (UM) and Analysis Packages. The User Monitor collects data on user performance and provides the User Profile data used by other parts of the Agent (Tutor and CLP). Analysis programs process user performance data to test hypotheses and modify the User Profile.

Tutor. This provides intelligent help to on-line users by explaining commands, reporting errors, introducing new features, and providing reference documentation. Tutor operation is also affected by the contents of the User Profile.

The Agent is designed to collect specific data about the user's use of the service, to make certain analyses of that data and, on the basis of the results, to recommend changes in the way the user deals with the service or the way the service looks to him. After we have gained actual user experience, we fully expect to have to change the nature of the data collected, the way it is structured, and how it is analyzed. In this process, however, we expect to learn a great deal about the critical parameters of a man-machine interface and how to control them to maximize the user's performance and satisfaction.

Reliability

Resilience of the service is an important aspect of a message service. The IA design calls for a distributed process across multiple host processors with redundant copies of the service's basic files dispersed among the hosts. If any one host is not operating, any user can then still be served. Since the processes are distributed, a user does not need to run on the machine which stores his files.

In order to make this work, file naming conventions must be coordinated to insure system-wide uniqueness. In the IA message service design there are three distributed processes, each of which controls a separate data base. The Coordination Daemon controls all messages in preparation; the Transmission Daemon controls all messages that have been released; and the User Daemon controls all user personal data files. Every host involved in the service has a copy of each of these daemons. When a user logs on, he is assigned to a host by the User Daemon. That host's daemons retrieve his personal files and then start up a job for him. This user job talks to the daemons for all its subsequent message file accesses. This distributed nature of the IA message service with redundant file storage provides the robustness required for a military environment. (Because of the cost implied by such a distributed service, this aspect of the IA Message Service design is not being implemented for the Oahu experiment.)

Security

Another important requirement for this message service is that it meet military security specifications. Although this test system will be operated at system high (i.e., all

users are cleared to the highest level the system will carry), with access restricted to TS personnel, it is a test objective that the message service address the security issue directly in its design principles. Previous research done by MITRE has identified the attributes that a system must possess to meet this criterion. The challenge is to build the service in such a way that someone can verify that the program indeed does have these attributes. The current state of program verification will not handle programs of the size and complexity of the IA system.

The approach being taken is to concentrate all of the security-relevant code in a single, small module (a security kernel). If it can be shown that this kernel does indeed handle all the security issues, the rest of the code does not need to be verified. The project schedule does not have time to actually verify this kernel, but the system is being designed with a kernel in it, which--given the proper effort--could be verified. The TENEX operating system, on which the message experiment will be run, is being altered to reflect military security, and is assumed to be a part of the kernel.

Privacy (discretionary control of message access on criteria other than security level) is another major concern in a message service. The principal difficulty here is in eliciting from the military a reasonable statement of what the rules should be. "Need to know" is a highly judgmental quality and very difficult to model. The IA project plans to embody access control mechanisms general enough to be applied to a broad set of models. The service will support author-assigned access control to annotations associated with messages and to personal files.

Scalability

In the COTCO study it was learned that during the average day on Oahu, 6,000 formal AUTODIN messages are sent out and 15,000 messages received. To insure that received messages get to the appropriate people an average of 40 copies are distributed. The CINCPAC communication center devotes a 24-hour-a-day printing press to this function. To handle traffic of this magnitude in an on-line system, it is necessary to organize the messages as efficiently as possible; for example, when a message is "delivered," instead of making a private copy for each recipient (as is done with current ARPA message services), the IA system delivers a brief "citation" to the message. The user is then granted read-only access to these central copies when he wishes to read its contents.

Other design decisions in the IA message service also reflect this concern for scalability. The organization of user files is also done by a central process (User Daemon) to compact them as much as possible. In this way, data relevant to many users can be kept in the same TENEX directory rather than requiring a directory per user. The daemons are distributed processes that operate across multiple hosts on the network so that the service can grow in a straightforward way by expanding the subnet (more nodes and more links) and adding more message processors.

Terminals

A significant part of the IA project is devoted to providing good human engineering to the military message service. For the traditional action officer to accept it, the service must be easy and simple to use; the cutting edge of this problem is his interaction at the CRT terminal. We believe it is therefore critical to provide two-dimensional editing with instantaneous feedback of trivial operations, as though the user's keystrokes actually performed the operation (insert a character, delete a character, move cursor, scroll, identify a cursor location as being data of interest, etc.). Other more complex operations are dealt with as commands to the system Agent. For these, indication that the command has been input should be instantaneous, but longer delay in actual performance of the action is acceptable.

For a message service test in which users are separated from the supporting host by a network, it is difficult to supply the necessary speed of response unless processing is done at the terminal site. With this two-stage architecture in mind, IA has designed a front end process (called Terminal Control) as a separable module for handling actions local to the terminal. Terminal Control will be put into a microprocessor which is to be provided as part of each CRT terminal on Oahu (see the final subsection of Section 7 for details of this development).

The Military Message Experiment

As described earlier, the principal focus of the IA project is on the test to be run at CINCPAC Headquarters in 1977. There are currently two other message services (being built by BBN and MIT) addressing this same test, although only one service will eventually be used for the operational test. In addition there is a PDP-10, communication equipment, and terminals to be procured, installed, and maintained. For cost considerations a team from MITRE and NRL (this will be a single-host test) is overseeing the security designs of each message service, while another individual is working on training requirements. A different group from MITRE is developing a Test Plan for the experiment, including the determination of data collection requirements and analysis to be conducted.

As a part of the Test Plan development MITRE has generated a Concept of Operations, which describes in detail the operation of the specific community of users that has been selected for this test, Operations Directorate (J3).

PROGRESS TO DATE

During the past year progress has been made on four fronts: understanding the user and his environment better, developing and presenting the plan for the Oahu

experiment, developing the software for the IA message service, and developing the terminal for this test.

Understanding the User

As this test has been slowly taking shape, the selection of the ultimate user community has been narrowing, until late this spring the J3 staff at CINCPAC was selected. MITRE personnel have spent time observing the operation of this group and have fairly well identified the operations these users currently perform in their message handling. From these, MITRE has abstracted a set of functions which the message service should support. Some of these are general-purpose in nature, while others are highly specific to this community. The general functions called for match quite well those stated as design objectives of the IA design in various earlier documents [5, 2, 6].

Another aspect of understanding the user involves providing the optimum language for him to communicate with the system. In 1974 [3] a methodology was outlined for selecting and improving this language. This methodology (which we call protocol analysis) was tested in 1975 [7] using a selection of computer scientists as subjects. In February 1976, a protocol analysis test was conducted in Washington D.C. using 21 subjects from various naval commands (NAVELEX, NAVTELCOM, NAVCOMUNITWASH). These subjects included senior naval officers, engineers, enlisted personnel and secretaries. The results of this test were published in May [8] and revealed interesting valuable data. The test served to ascertain how the users wanted to be able to use the system, how they wished to phrase instructions to the system, which functions they used most frequently, what vocabulary they employed, and so forth. It also helped in identifying many necessary functional requirements of the service (e.g., by clarifying the overall structure of the message operation and the interaction of its parts). Since the nature of formal communications within NAVELEX and NAVTELCOM appears to be different from that in the Operations staff at CINCPAC, in July of this year a protocol analysis will be conducted on Oahu using a representative sample from the eventual community of users in order to understand their specific needs.

Developing the Plan for the Experiment

Since ISI has been involved in this ARPA program from its inception, it has participated to some degree in many phases of its planning. In January 1975, the IA project produced an informal report to ARPA and NAVELEX called *Military Message Processing System Design* which outlines our ideas of how a test plan should be organized and what it might contain. Since then MITRE has been contracted to develop a formal test plan, and the IA project has reviewed and critiqued the various drafts of that document as it has evolved.

In order to assist ARPA and NAVELEX in planning for this experiment, the project has produced a number of reports and documents during the past year, including a recommended TENEX configuration, a study of potential systems for use as the data concentrator for the test, a specification of the functions and equipment required for that data concentrator, and several PERT charts for the system integration for the experiment. The recent attention to security has required IA participation in joint discussions with the other message service developers and the security control team from MITRE and NRL. From these discussions each service is developing its own approach to handling security.

Message Service Development

During the past year the IA message service has gone from a design on paper to a near operational system. From the start the service has been divided into two major phases. Phase 1 involved development of the Agent and the creation and coordination aspects of the message service; this has required bringing up at least skeletal versions of all the basic modules of the service. Phase 2 covers the message reception, archival, and retrieval features of the service. Phase 1 is nearly complete, and Phase 2 is under way.

The parts of the Agent developed this year include the Terminal Simulator, the Command Language Processor, and part of the Tutor. The terminal simulator is code-resident in the PDP-10, which is designed to make the HP 2640A look to the user and the rest of the service like the new terminal. It provides multiple windows, with independent scrolling, local editing, and function keys. This simulator is necessary until the new terminal is ready.

The Command Language Processor (CLP) does the command interpretation and makes appropriate calls on the functional module. The CLP is table-driven, making the addition of new or altered commands straightforward. The CLP controls a two-line "command window" on the screen; it provides spelling correction and automatic completion of commands and arguments. All lexemes handled are compared to a synonym lexicon before being parsed, allowing flexible substitution of values.

A help facility is being developed as the first part of the Tutor. It operates on syntactical entities (terms the user wants to know about). A menu is provided from which the user selects the nature of information he wants and the level of "verbosity" of the answer.

The functional aspects of message creation and coordination are provided by the Functional Module and the Message Access Module of the user job and the Coordination Daemon. The Access Module deals with the message as it is stored on file by the system. The Functional Module converts the commands from the CLP into appropriate calls on the Access Module and displays the results on the terminal, through a submodule called the Virtual Terminal. The Access Module shields the Functional Module from the detailed file

representation of the structure of a message. The Virtual Terminal shields the Functional Module from details of the terminal characteristics.

The VT, FM, and Access Module currently support creation and coordination of a message. An originator drafts a message, adds comments to fields as desired, fills in the Coordination field with user names, and then sends it for coordination. The message coordinator may then edit the message and add comments of his own. In addition, he may retrieve lower level information such as precedence assigned to the message review, and whether he has been asked to sign off or just edit and comment on the message. A coordinator may start a sub-coordination list of his own. The author may then compare coordinators' renditions with his original, or with each other, via split screen display.

The Functional Module is associated with each logged-on user. The Coordination Daemon controls the central messages-in-progress file, where all messages being created or in coordination reside. When a drafter originates a message, or a coordinator adds his changes and comments, this process controls writing the update into the actual message file itself. In addition the daemon sends out citations to users when they are due to be notified of a message ready for review. Currently the Coordination Daemon allows just a single user access to a message at one time. Although it generates citations for messages, the reception phase of the system cannot handle them beyond alerting the user of their arrival.

Message Transmission

When a message has been reviewed appropriately it is released for transmissions. The Coordination Daemon prunes off the excess renditions, all comments, and the coordination list, and passes the message to the Transmission Daemon. This process is responsible for formatting and sending the resultant message to its destination. Eventually for the message experiment this will be the LDMX and AUTODIN. At this time the message is formatted as an ARPANET message and given to the TENEX Mailer process.

Messages received from the LDMX will likewise be processed by the Transmission Daemon and converted to the IA internal form. Recipients will then be sent citations for these messages and the Functional Module of user jobs accesses the message in a similar manner as a message in coordination. Currently ARPANET messages are processed in this manner.

Message Reception

Message reception on the IA system centers around the concept of file folders. Initial citations to messages arrive from the daemons and are placed into a "Pending" folder, somewhat like a person's in basket or mail box. From the Pending file, messages

may be moved into personal folders or to more generally accessible folders--analogous to read boards or bulletin boards. Although the user thinks in terms of folders holding messages, in fact only limited message data is stored. From this limited data, the full message may be directly retrieved.

The handling of folders is done much like message handling--through a special Folder Access Module that isolates the Functional Module from having to know much about the internals of this file representation. The Folder Access Module is being coded at this time.

Folders have a number of special fields which dictate how their entries are handled. One field is an automatic input filter which allows messages to be automatically filed without explicit user intervention. A Template field specifies what fields of the message are stored for this particular folder and what fields or parts of fields are displayed when the contents are examined. Thus the folder may store key words and the To field of each message, but not necessarily display these to the user. These fields can then be used for retrieval.

CONTINUING WORK

The IA project has a goal of having an operational message service, including preparation and reception phases, by the end of November 1976. A set of functional requirements have been established by the Test Director for the Military Message Experiment. The IA message service will meet the majority of these requirements by that time.

During the subsequent four months, the message service will be shaken down using the Oahu host computer, live LDMX traffic, the new CRT terminals, and friendly military users. During this period user documentation will be completed and test data collection routines will be incorporated into the service. Additional terminals will be built and delivered.

In the spring of 1977, the emphasis of the project will shift from development and shakedown to training, evaluation, and upgrading. It is anticipated that a number of specific "structured" tests will be conducted to evaluate specific features of the service. During this period it is anticipated that an ISI representative will be on-site with the users to assist in training and tailoring the service to the users' needs.

By approximately July 1977 the formal operational message service test will begin. By this time the principal focus of the project will be on user and system support and on data collection and analysis. Since this will be the first time unbiased users will be operating the system, this will be a primary opportunity to study the effectiveness of the agent in supporting the nonspecialist user.

REFERENCES

1. Ellis, T. O., L. Gallenson, J. F. Heafner, and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, ISI/RR-73-12, May 1973.

2. Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, ISI/RR-74-26, May 1975.

3. Heafner, J. F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, ISI/RR-74-21, September 1974.

4. Abbott, R. J., *A Command Language Processor for Flexible Interface Design*, ISI/RR-74-24, September 1974.

5. Tugender, R., and D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, ISI/RR-74-23, May 1975.

6. Rothenberg, J. G., *An Editor to Support Military Message Processing Personnel*, ISI/RR-74-27, June 1975.

7. Heafner, J. F., *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, ISI/RR-75-34, July 1975.

8. Heafner, J. F., M. D. Yonke, and J. G. Rothenberg, *Design Considerations for a Computerized Message Service Based on Washington, D.C., Navy Personnel*, ISI/WP-1, May 1976.

6.
NETWORK SECURE COMMUNICATION

Research Staff:
Danny Cohen
Thomas L. Boynton
Stephen L. Casner
E. Randolph Cole
James Koda
Eric Mader
Robert Parker
Paul Raveling

Research Assistant:
John Kastner

Support Staff:
Nancy Dechter
George Dietrich
Oralio Garza
Clarence Perkins
Leo Yamanaka

INTRODUCTION

Modern military command and control techniques have created a critical need for secure, low-bandwidth voice communication systems which maintain high speech quality, operate in real time, and permit full duplex (simultaneous in both directions) communications. Such systems should ultimately provide the capability for conferences between many users at multiple sites, with an efficient means for controlling the conference. If these systems are to be fully secured, digital communication techniques are necessary and must be developed.

Another trend in military communications is the use of packet-switched computer networks, such as AUTODIN II, for data communications. Beginning in the 1980s, a large portion of the military computer communications load will be handled by packet-switched networks, made up of telephone, radio, and satellite links. A capability for secure voice communications over packet-switched networks would provide an efficient, cost-effective response to much of the secure voice communications problem, including conferencing. The high ratio between peak and average data rates for voice communication makes a packet-switched network an ideal communications medium.

A primary objective of the ARPA Network Secure Communication (NSC) effort is to demonstrate the feasibility of secure, high-quality, low-bandwidth, full-duplex digital voice communications over packet-switched computer communications networks. Much of this objective has been accomplished using the ARPANET, which has been the model for both military and commercial packet-switched networks.

BACKGROUND

The ARPA NSC effort has been in progress since late 1973. The initial tasks were to specify a high-quality low-bandwidth speech compression algorithm, select a high-speed signal processing computer which could execute the algorithm in real time, and select a host and operating system to interface the processor to the ARPANET.

Linear Predictive Coding (LPC) was selected as the high-quality low-bandwidth speech compression technique because it seemed to represent the best tradeoff between computational complexity, bandwidth, and quality. The specific algorithm chosen was a Markel autocorrelation-type LPC of order 10 using SIFT (Simple Inverse Filtering Tracking) pitch extraction [MARKEL 72], [MARKEL 74]. The LPC vocoder extracts 12 parameters from a 9.6-millisecond frame of speech: pitch, gain, and 10 k-parameters (often called reflection coefficients). The 12 parameters from each frame are encoded into 67 bits. About 52 frames per second are transmitted, giving a transmitted bit rate of about 3500 bits per second. Nothing is transmitted during periods when the speaker is silent. This system, called the Phase I system, is a fixed-rate, fixed-order system; it is full-duplex (simultaneous analysis and synthesis of speech).

In 1973 a complete survey of high-speed signal processing computers was made, and the Signal Processing Systems SPS-41 was selected to be used by the NSC group. The SPS-41 is a 16-bit integer machine capable of 4 million 16-bit integer multiplications per second, but with limited program and data storage and wire-wrapped construction.

The PDP-11 was chosen as the host for the LPC system, running under the ELF operating system developed at Speech Communications Research Laboratory (SCRL). The PDP-11 is to act as a host on the ARPANET and control the assembly and disassembly of packets, input and output to and from the SPS-41, and the operation of the SPS-41 itself. It should be noted that not all NSC group members used the SPS-41 or PDP-11; Lincoln Laboratory first used the TX-2 host and FDP signal processor, and later used a PDP-11 with a Lincoln Digital Voice Terminal signal processor, while Culler-Harrison, Inc. (CHI) used two machines of their own design.

With the hardware chosen and the algorithm defined, the next step was twofold: formulation of a Network Voice Protocol (NVP) and implementation of the LPC system on the SPS-41/PDP-11 combination. Preliminary specifications for the NVP were issued in June 1974 and final specifications in October 1974. Since implementing the LPC system would be a time-consuming task, it was decided to test the NVP initially using CVSD (Continuously Variable Slope Delta Modulation, the DOD "standard" high-rate vocoder), which is particularly easy to implement. Therefore, in September 1974, an 8-kilobit CVSD system was used to test early versions of the NVP; this greatly aided in the operational development of LPC, revealing several problems with higher-than-usual data rates on the ARPANET which were quickly corrected. The first CVSD tests were held between ISI (using the SPS-41/PDP-11 system) and Lincoln Laboratory (using their FDP/TX-2 system). This was a critical test for the NVP, since the systems were quite different.

After encountering many SPS-41 design problems, ISI brought up LPC on the SPS-41/PDP-11 combination in March 1974. In January 1976 the first LPC tests were run on the ARPANET, between Lincoln Laboratory and CHI. Again the hardware on both ends was completely different. The ARPA LPC system met all its specifications and demonstrated that the combination of low-bit-rate speech and a packet-switched computer network was an effective one.

The next step in demonstrating the usefulness of digital speech on packet-switched networks was to implement a conferencing system. To do this it was first necessary to expand the NVP into a Network Voice Conferencing Protocol, or NVCP [COHEN]. This was done, and on January 23, 1976, a four-way digital conference was held among ISI, Lincoln Laboratory, CHI, and SRI, using the standard Phase I LPC algorithm. SRI used software written by ISI and modified slightly for their system, with ISI's help.

In 1975, in parallel with the network LPC conferencing system, ISI developed a sophisticated local CVSD conferencing system, called MA-BELL, for experiments in practical digital conferencing within the Institute. In addition, MA-BELL is serving as a development vehicle for the development of a convenient, human-oriented user interface for digital conferencing systems. Future plans include expansion of the CVSD conferencing so that participants can also be connected via the ARPANET.

In all of this, ISI's NSC project has acted in the role of coordinator among the contractors. The NVP and the NVCP were originally drawn up at ISI, then modified into final form in discussions with the other ARPA NSC sites. The SPS-41 LPC analysis and all the PDP-11 support software were written at ISI.

Acknowledgments

Throughout the ARPA NSC effort, the cooperation between the NSC sites has been truly exceptional, as has been the guidance of the ARPA program management. Progams and hardware designs have been freely exchanged. For example, the SPS LPC analysis was written at ISI, except for the matrix solution subroutine SOLVE, which was written by BBN, and synthesis was written at SRI. Support software for the network voice system came from ISI, BBN, SCRL, and SRI. Technical consulting came from SCRL, BBN, Lincoln, and Utah. Lincoln, CHI, and SRI cooperated fully in bringing up the system.

APPROACH

The primary objective of ISI's approach to Network Secure Communications has been to develop systems and techniques which are as generalized as possible. The NSC low-bandwidth packet speech system is not specific to any one vocoder, such as CVSD or LPC, or to any one packet-switched network, such as the ARPANET. Of course, some portions of the system must be specific to the vocoder or the network in use; however, these portions are either encapsulated in modules, such as in the LPC vocoder drivers, or could be easily adapted, as in the network control routines.

The ISI NSC project has implemented a conferencing system, called MA-BELL, which is presently based on CVSD and operates locally between several offices at ISI. In the future it will be expanded into a transnetwork conferencing system. Figure 6.1 illustrates a conference with four users, one of whom acts as the chairman. The system control panel is shown in the center of the figure, with three users using CVSD vocoders plus a control box, and the fourth using a standard pushbutton telephone.

MA-BELL can handle several participants, in either a point-to-point or a conferencing mode. Each user may communicate via a CVSD vocoder (three are shown in Figure 6.1) and a control box (see Figure 6.2), or via any ordinary telephone equipped with a pushbutton pad. In the latter case, the phone's pushbuttons serve as the control panel. The purpose of the control panel for the ordinary participant is to enter or leave the conference, ask for or relinquish the floor, enter a vote, etc. The chairman's own control panel, in addition to those functions, can control the entire conference, assigning or reassigning the floor, inviting participants, initiating a vote, etc.

The chairman can also control the conference from the system control panel, SCP. The SCP shows all the available information about the participants, including a graphic display of the connections between the participants, who has the floor, which participants have expressed a wish to talk (and are queued), whether the current speaker has been warned that he is about to lose the floor to the next speaker, status of open votes, and other functions.

Conferences are initiated by the chairman, who issues the conference-ID and the list of parties allowed to participate. Thereafter each participant may join by "dialing" the right sequence, which includes the conference-ID. As long as there are only two participants in a conference (the chairman and the first "joiner"), no conference discipline is enforced and both can talk to each other just as they could over a regular point-to-point connection; when the third participant joins in, however, the conference discipline is enforced and the participants can be heard only when they have the floor.

The floor assignment is either manual--by the chairman--or automatic. The automatic schedule, which is currently that most frequently used, takes the floor away from the current speaker 15 seconds after a request is made by another user of equal or higher priority. The current speaker is warned 5 seconds before he loses the floor. Inside ISI the participants usually have the same priority. The "behavior" of the "scheduler" can easily be modified by changing the parameters involved and the priorities assigned to each participant.

Control output to the participants (like "you may speak now," "you have lost the floor" etc.) are issued both by light signals to the control boxes and also vocally (by use of prerecorded messages). Controlling a conference can be done better and more easily from the SCP than from any other station, because all the information about the state of the conference is graphically displayed in an easy-to-read and axiomatic fashion.

Key features of this approach to packet speech are

- Separation of control messages from data messages.
- Robust performance in the event of lost messages.
- Elimination of the possibility of deadlocks.
- No end-to-end retransmission of data.
- Dynamic adaptation to changing network performance.
- Sufficient guaranteed bandwidth for speech with minimum delay.
- Support of a flexible, human-oriented user interface.

All of these approach guidelines have been and are being followed throughout the NSC project, although a few are very difficult to achieve in actual practice. For example, dynamic adaptation to changing network performance will require additional work in the network measurements area, leading to better short-range measurements, before a truly dynamic packet speech system can be achieved.

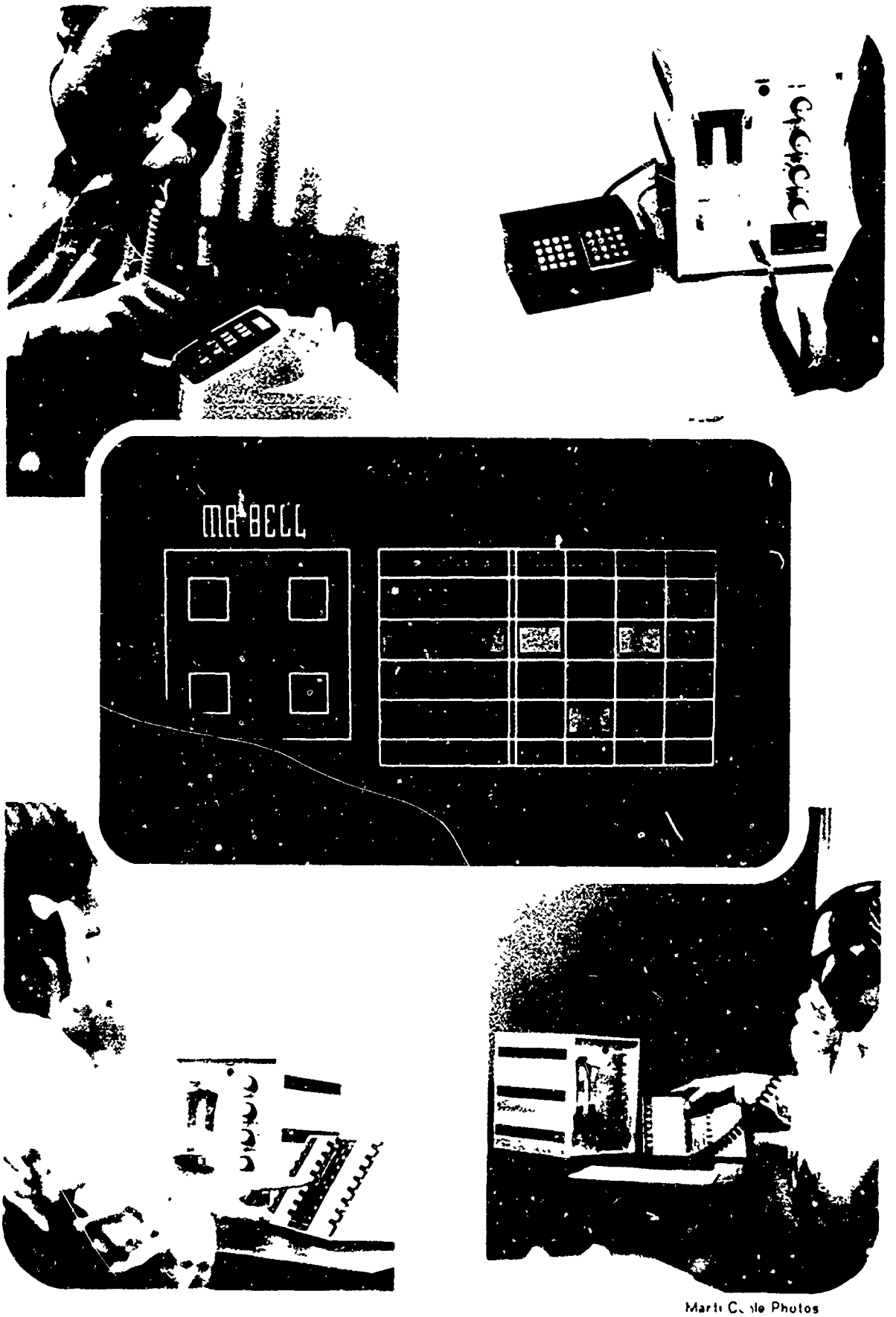
A similar approach has been used in the choice and implementation of the vocoders. The most important consideration was the highest possible speech quality at relatively low bandwidth, which led to the choice of LPC. While the present Phase I LPC system operates with a fixed output rate whenever the speaker is speaking, the Phase II LPC system currently being implemented operates with a variable output rate depending on the speech itself. The latter system will exhibit a higher peak rate with a considerably lower average rate than the former, a feature which is particularly compatible with a packet-switched network.

In addition, the CVSD vocoding technique was also used, not because of its features, but because CVSD was chosen by the DOD as its standard "high rate" vocoding system.

PROGRESS

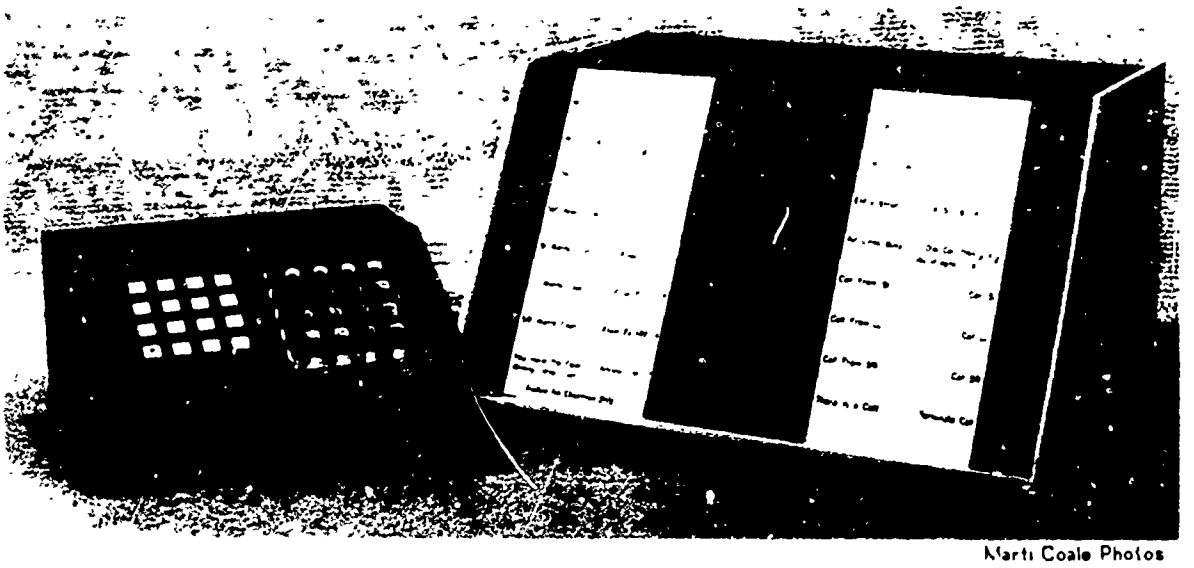
Conferencing System

Most vocoders in use are not linear in the sense that the output bit streams cannot be added in any meaningful sense. Therefore, a participant with only one vocoder can listen to only one speaker at a time. This does not allow the "common air" as in a normal phone conference, and creates a need for conference control. Therefore, the central issue in conferencing is the control of voice data flow and of control information flow.



Marti C. le Photos

Figure 6.1 A conference using the ISI conferencing system



Marti Coale Photos

Figure 6.2 Two generations of control boxes

The chairman has two main functions: scheduling (floor control) and directing the speaker by talking privately to him wh e he speaks. The chairman has an open line to the speaker at all times. This allows the chairman to guide the speaker with comments like "Please summarize your position, since I am about to take the floor from you."

The voting process allows a user to vote (yes or no) without being polled or having the floor. This is useful not only for voting but also for questions put forth on the floor like "any comments?", "any questions?", etc., since it saves the tedious polling which is otherwise necessary.

In the future, the chairman will be able to direct that the conference or portions of the conference be recorded. This will provide an on-line conference transcript, which acts like any other conference participant. All participants in the conference will be informed that the conference is being recorded. Additional safeguards, such as allowing transcription only after a unanimous vote of the participants, could also be implemented.

The Network Voice Conference Protocol (NVCP)

The conferencing feature of NVCP is based on the following model:

- Each participant has only one vocoder and therefore can listen to (only) one speaker at a time.

- In each "station" (i.e., host) participating in the conference, there is a process controlling the access of all the local individuals ("extensions," "participants") involved in the conference. This process is called the "local conference controller" (LCC).

- There is one conference chairman, either one of the participants or a program (co-located with one of the participants), which decides who is listening to whom and when.

- All the conference handling is in addition to the regular NVP procedures.

The Network Voice Conference Protocol is only a control protocol, making use of the same data protocols as used by the NVP. In fact, NVCP is defined as an extension to the NVP control protocol.

It is most important to realize that the NVCP per se does not address the issue of the man/machine communication either between the participants and the LCC, or between the human chairing the conference, if any, and the CHAIRMAN program controlling the conference. This issue is an implementation issue and does not belong to the protocol; however, some recommendations for the man/machine interface are included with the NVP protocol.

The conference structure. During a conference two logical networks exist: the first is a high-bandwidth network carrying the voice data from the speaker to all the other participants; the other is a low-bandwidth network carrying control information between each participant and the conference chairman.

The first logical network, the data (or voice) network, is dynamically modified as the different users become (and cease to be) the speakers. Whenever a participant receives the floor (i.e., becomes the new speaker), the data network is reconfigured to allow data to flow from this participant to all the others.

In contrast to the data network, the control network has a static structure, since the conference chairman does not change during the duration of the conference.

Two alternatives were examined and compared: one was to treat each individual participant separately from the others in the same site; the other was to group all participants in the same site for optimizing of the network utilization. No advantage, except simplification of some programs, seems to justify the first possibility, and the advantage in optimizing network utilization resulted in the decision to group all participants at the same site together.

Two possibilities were considered: negotiation with the chairman once per participant, or once per host. The latter was chosen for better efficiency.

Two data traffic patterns were compared, one from the current speaker to all other sites, and the second from the speaker to the chairman, then to be distributed to all sites. The first provides better network utilization and smaller delays. The second is simpler, since the data network does not change during the conference as required by the first method. Again, it was decided to sacrifice a little simplicity for the sake of network utilization.

The user interface supported by the NVCP. The NVCP can support a user interface which allows the user to do the following:

- Define a conference (with optional participants list) to be chaired by this user, or by any other participant at the discretion of this user.
- Join a conference chaired by another user (dialing).
- Request the floor.
- Relinquish the floor.
- Perform several functions whose meaning is defined at conference time (e.g., voting, "yes" and "no").
- Terminate his participation.

The user interface will also allow the system to tell the participant

- That he is "in"/"out" of a conference.
- That he may or may not talk.
- Several functions to be assigned at conference time (e.g., "You have two minutes to finish talking").

The conference organization. A conference starts when the chairman tells his system that he wants to initiate a conference and defines the participants list. At this stage, a conference chairman control program is initiated.

Whenever a user wants to participate in a conference, he contacts the chairman (via the LCC) and discusses it with him (using the NVP negotiation procedure). Upon acceptance of this user into the conference, the chairman contacts the LCC, telling it to add this user to the participants list. Any active LCC is always told by the chairman from which host data should arrive, on the link assigned by this LCC (in the initial call).

When a change of speaker occurs, first the chairman tells the speaker that he does not have the floor, then all the LCCs are set to receive data from the host of the new speaker, and finally the host of the new speaker is told to send data to all the participating hosts. Only after the LCC finds itself prepared to ship the data out is the user notified to start talking. (Of course, change of speakers in the same site may be simpler, since other hosts do not have to change their receiving procedure.) At any time each participant may send to the chairman control information (e.g., "I want to talk next"). The chairman may at any time send control information to any participant.

As a general principle, all control messages are between the chairman and the LCCs, never directly between the chairman and the participants themselves. However, the NVCP provides a means for direct control communication between the chairman and each participant. This information is communicated via 8-bit bytes whose meaning is not interpreted by the LCC. This communication is carried by special control messages (e.g., "I want to talk"). Control information between the chairman and the individual participants that has to be understood by the LCC is carried by other messages (e.g., "please shut up").

Comments. Note that the chairman might never speak if he so wishes (e.g., when it is an automatic program). There is no requirement for the chairman ever to send data messages (i.e., voice).

The chairman may assume that all his instructions to the LCCs are followed at once. It is the responsibility of the LCCs which delay their action (probably depending on data flow) to remain synchronized.

NVCP, like NVP, uses the controlled messages (Type 0/0) for all control information, and might use uncontrolled (Type 0/3) or "normal" messages for voice data. Each vocoder is expected to start its output with the time-stamp (i.e., parcel number) set to zero.

Summary. With the model upon which our system is based, it can be recognized that a protocol is needed. The NVCP provides the needed facilities for a real-time packet-switched network conference system.

The EPOS Operating System

One of the critical links in the network/host/signal processor chain which is necessary for packet speech is the host and its operating system. The ELF operating system, developed by SCRL, was originally chosen for the NSC project's PDP-11 operating system because it was originally tailored for speech applications and was the only available PDP-11 operating system capable of supporting packet speech.

Both the initial CVSD and LPC packet speech systems were implemented using ELF as the PDP-11 operating system. During the network experiments with these systems,

however, it was discovered that the PDP-11 was very heavily loaded and that most of the end-to-end delay was caused not by the ARPANET, but by delays within ELF itself. For example, speech packets were recopied from buffer to buffer in the PDP-11 several times before being processed, each time causing additional delay and overhead. The PDP-11 was busy from 90 to 100 percent of the time handling one 3500 bps LPC vocoder, as measured by UNIBUS utilization. The 8000 bps rate of the initial CVSD-based packet speech experiments was handled by an early, non-virtual-memory version of ELF which was not powerful enough to handle the LPC system. Therefore, in order to handle LPC conferencing, transnet and local CVSD conferencing, and possible future applications, the operating system was re-designed. The result, EPOS, is the Environment for Processing of On-line Speech.

Although the differences between EPOS and ELF are extensive, some of the major speed improvements were accomplished by:

- Restructuring system control blocks to optimize the code referencing them; in particular, the context switch code was reduced by half.

- Reducing the number of context switches required in interval timer handling by eliminating the process which managed the interval queue, again reducing the overhead by more than one half.

- Using a more efficient algorithm for inter-address-space data transfers, cutting the time by a factor of 8 to 20.

- Totally re-organizing the I/O architecture and control structures. Rather than having a general I/O control process with drivers for specific devices, EPOS uses a separate process for each device; this allows the code to be specialized for each device, so that fewer instructions are executed than for a generalized process. I/O operations pass less information because more is kept in kernel control blocks. This allows passing the parameters in registers rather than copying a block of information to the kernel, cutting the overhead by a factor of five.

- The network control process takes full advantage of the capabilities of the IMP interface to provide maximum throughput. Specifically, for the network speech applications, network messages are input and output to/from the user program's buffers, eliminating expensive copying. Probably the most significant reduction in overhead is due to the efficient management of network I/O.

In addition to performance improvements, EPOS has a number of features which make it easier to use:

- System control blocks are not statically allocated; rather they are allocated from dynamic storage as they are needed. This means that the system need not be tuned for the resource requirements of a specific application at system generation time.

- The concept of a "job" is incorporated into the kernel of EPOS, rather than the EXEC as in ELF. Putting user programs in separate jobs allows isolation from errant neighbors and facilitates protecting the system against user errors.

- I/O calls in EPOS are like those of TENEX. We have found that the ease of use of these calls has significantly reduced the time required to write application programs.

In more detail, EPOS supplies the following facilities for application programs:

- Process management: EPOS combines process scheduling and interprocess communication in a single signal/wait mechanism. Semaphore operations are also available. The system supports all normal auxilliary functions (process creation, deletion, freezing, and thawing). This allows segmentation of application programs into muitiple processes to correspond to the logical structure of the task.

- Job Management: A job is an independent computing environment, each with its own set of private resources such as address spaces, processes, open files, and assigned I/O devices. The kernel's address space and kernel processes belong to job 1, which is allowed special privileges by several system functions. Other jobs run in user or supervisor space and are created in response to a control-C input from any unassigned terminal.

- Memory management: EPOS allows each job to use multiple address spaces and to use separate instruction and data spaces on machines whose memory management hardware supports this feature. "Virtual move" functions allow transfer of data between address spaces within a user job, or (when invoked by kernel functions) between any address spaces. Page mapping allows sharing memory pages and including particular physical pages in an address space. The latter feature is used primarily for communication with signal processors through shared memory.

- I/O management: System calls handle either bytes or strings. The latter may have a specific length or may be delimited by a specified character, such as a 0 byte. EPOS also supplies asynchronous I/O calls, which do not block the calling

process while its request is pending or in progress. Devices currently supported include terminals with either DL11 or DJ11 interfaces, IMP interfaces, disks, and MX-521 CVSD vocoders. Disk I/O is currently limited to reading and writing existing DOS files. Files opened on a pseudo-device (IPP:) become interprocess ports; this is the only means of communication between user jobs which does not require doing physical I/O.

- Special-purpose hardware support: In addition to page mapping, several system calls allow user processes to perform device manipulation functions for signal processors or other special-purpose hardware. User processes can allocate interrupt vectors and specify the contents of signals to be issued when the interrupts occur, and they can get and set the contents of particular device interface registers.

Along with the EPOS operating system, the EPOS Exec provides a convenient environment for debugging and using application programs. The Exec has external characteristics almost identical to the TENEX Exec. It provides utility services for loading and running programs and checking their status. It also communicates with the debugger to allow a user to change easily to debugging mode if a program requires modification or repair.

One important feature of any programming environment is a good debugger. All the existing PDP-11 debuggers of which we were aware fell far short of our standard, TENEX IDDT. Many did not provide any symbolic capabilities at all (user-defined symbols, instruction type-in, etc.). Most were written to run stand-alone on a PDP-11 and would have required extensive modifications to function in the EPOS multi-process environment.

Because of these problems and because we felt that a powerful debugger was essential to enable us to produce good software efficiently, we designed MEND, the Multi Environment Native Debugger. The design provided the most powerful set of features provided by any debugger with which we were familiar (including IDDT). Implementation of MEND has been approximately 75 percent completed; the only major feature lacking is the breakpoint and trace facility, upon which implementation has already begun. Debugging done with the partially completed MEND has confirmed that we have indeed provided a most powerful debugging tool.

High-Speed Signal Processors

Much of the effort spent in the LPC implementation portion of the NSC project was spent in isolating and correcting design problems in the SPS-41 signal processor which prevented the LPC programs from running. The Phase I version of LPC consumes about 95 percent of the SPS-41's time, and uses 13 of the machine's 16 Input/Output Processor channels. Considerable time and effort was spent in adapting algorithms to the

fixed-point architecture of the SPS. Even after many problems were corrected, the SPS-41 was not reliable enough for useful demonstration work; therefore it became apparent that, in order to do high-quality packet speech research and demonstrate the results, a more powerful, reliable, easier-to-program processor was required. Consequently, in late 1975, an evaluation of the available state-of-the-art high speed signal processors was undertaken. The requirements which had to be met were as follows:

- High reliability.
- Expandability.
- Availability of a simulator written in a high-level language and runnable on TENEX.
- Portability.
- Floating point arithmetic.

More than 10 machines were surveyed, mostly commercial products, with some research-type machines. Cost, availability, and capabilities soon narrowed the field to two machines. An effort was made to obtain simulators (runnable on TENEX) for both machines and write and run programs on them to assess the relative difficulty of programming each machine. A secondary goal was to estimate the size and speed of LPC running on both machines. ISI was able to obtain the simulator for one machine, and implemented the basic signal-processing subroutines needed for LPC on that simulator in a few weeks. That machine, the Floating Point Systems AP-120B (FPS) was chosen as the most suitable signal processing system.

The FPS AP-120B machine was delivered to ISI in June 1976, and interfaced to the NSC project's PDP-11/45. The Phase I LPC system is expected to be operational on the FPS/PDP-11 combination in July 1976, with the Phase II system to follow a month later. ISI's experience to date indicates that the FPS is reliable and programs which run on the simulator will run on the FPS.

LPC

Most of the progress in the area of Linear Predictive Coding (LPC) in the past year has been in LPC conferencing and the acquisition of a new, reliable, high-speed signal processing computer (the FPS) and the implementation of LPC. The original transnetwork LPC experiments were conducted in late FY75, and the operating Phase I LPC algorithm has changed little since then, except for some relatively minor quality improvements.

The major effort in LPC conferencing involved the development of a Network Voice Conferencing Protocol (NVCP), which is described in [COHEN]. The conferencing system was brought up in December 1975, and uses the standard Phase I LPC. When the Phase II LPC is implemented, it can be easily installed in the conferencing system because of the modularity and structure of the NVCP.

A Phase II LPC system has been specified by BBN [VISHU]. The Phase II LPC system will be a variable-rate system with a peak rate of about 5500 bps but an average rate of less than 2000 bps, compared with the constant 3500 bps output of the Phase I LPC system. Both rates apply to non-silence. Neither the Phase I nor the Phase II system transmits anything during silence periods exceeding about 200 ms. This is achieved by using a distance measure between the analysis parameters generated by the LPC vocoder. The Phase II vocoder operates at a fixed frame rate of about 100 frames/second, using the distance measure to compare each frame's parameters with the last set of parameters transmitted. The new parameters are transmitted only if they differ sufficiently from the previous ones to warrant transmission.

The Phase I LPC system has been implemented and is being tested on the FPS AP-120B signal processor. This implementation is a special case of Phase II. Within a short time the Phase I LPC system using the FPS will be operational on the ARPANET, followed shortly thereafter by the Phase II system.

Hardware Development

Hardware developed for MA-BELL includes

- An interface (the PB-11) which allows four CVSD vocoders to be interfaced to the PDP-11 through a single DR11-C word-at-a-time interface.
- A silence detection scheme (based on one developed by Lincoln Laboratory) was implemented as part of the PB-11, allowing the PDP-11 to tell when a speaker using a CVSD vocoder is silent.
- Two generations of control boxes for conference control and general-purpose use on the PDP-11 (see Figure 6). (Note the decrease in size from the first to the second generation.)
- A sophisticated telephone-answering and calling system which allows general-purpose input and output from a touch-tone telephone to and from the PDP-11.

RESEARCH AND DEVELOPMENT GOALS

The overall goal of the NSC project at ISI is to provide high-quality research and development in all areas concerned with digital voice transmission over packet-switched networks. Specific goals for future research and development activity are

- Further development of on-line conferencing systems.
- Development of dynamic on-line and off-line systems for storage, transmission, and retrieval of speech, including FTP and voice message capabilities.
- Integration of speaker authentication and possible isolated word or phrase recognition into all areas of the network speech systems.
- Continued attention to important issues of efficient, human-oriented user interfaces to network speech systems.
- Implementation and use of two reliable network speech demonstration systems based on the FPS AP-120B high-speed signal processing computer.

IMPACT

The NSC effort can be expected to have a broad impact on one high-priority military item, secure digital voice communication. In digital communications in general, the NSC work has provided the prototype of a packet-switched voice communications system. There is little doubt in anyone's mind that at some point in the future all speech communications will be transmitted digitally, whether it be military, business, or personal communication. Since packet switching has been proved to be a highly cost-effective and bandwidth-conserving technique, there will certainly be many future packet speech systems. Packet speech techniques can be applied to any digital transmission medium, whether it is telephone, radio, satellite, or optical.

There will be a need for well-engineered user interfaces for any and all future communications systems, an area which is often developed as an afterthought, particularly in areas other than personal and consumer applications.

Packet speech has already had an appreciable impact on networking. It was the first system on the ARPANET to require relatively high bandwidth with low delays without rigid error detection and/or correction. A new type of message, the Type 3 or "minimum-effort" message, was implemented as a direct result of this requirement. The packet speech system has brought improvements in dynamic network measurements at the host-to-host level, in order to optimize communications, and will bring about more improvements in dynamic network measurements in the future.

The signal processing aspects of the NSC effort have already had a wide impact on the speech compression and military communications communities. The ARPA NSC effort and the people involved in it have greatly influenced the low-cost low-bandwidth vocoder specified by the DOD consortium, of which ARPA was a member. The LPC algorithm is serving as a front end to authentication systems developed at SCRL and phrase recognition systems developed at Lincoln Laboratory. Using the NVP, these systems can and will be used via the ARPANET. There will also be significant impact on other areas, including operating system design for systems handling high-bandwidth data such as speech, practical low-cost high-speed computing systems using signal processors, and, of course, future network design.

REFERENCES

[MARKEL 72] Markel, John D., "The SIFT Algorithm for Fundamental Frequency Estimation," *IEEE Transactions on Audio and Electroacoustics*, Vol. AV-20, No. 5, December 1972, pp. 367-77.

[MARKEL 74] NSC Note 20, *Linear Prediction Analysis/Synthesis Program*, John D. Markel, Speech Communications Research Laboratory, Inc., 1974.

[COHEN] Cohen, Dan, *Specifications for the Network Voice Protocol*, USC/Information Sciences Institute, ISI/RR-75-39, March 1976.

[VISHU] NSC Note 82, *Specifications for ARPA-LPC System II*, R. Viswanathan and John Makhoul, Bolt Beranek & Newman, Inc.

7.
SPECIAL PROJECTS

XEROX GRAPHICS PRINTER

Research Staff:
Stephen D. Crocker
Pete Alfvin
Ronald Currier
Dono van-Mierop

Since its inception, ISI has undertaken several hardware development efforts in support of research requirements or to demonstrate a capability for a recognized DoD application. As reported in [AR 74], one of the most significant of these projects is the development and use of the Xerox Graphics Printer (XGP), a high-quality document printing capability in the form of a network terminal.

Two XGP systems have been installed, one at ARPA and one at ISI. They provide high-quality on-line hard copy with proportional spacing of characters according to width, and use of multiple fonts. This report is an example of the XGP's capabilities.

From mid-1974, when these systems were installed, until September 1975, the components of the XGP systems at both ARPA and ISI consisted of a modified Xerox machine interfaced to a PDP-11/40 with 32K words of core and 256K words of disk, interfaced via a 2400-baud line to the ARPA TIP, and driven over the ARPANET by any TENEX system, particularly OFFICE-1, ISI, and ISIB. See Figure 7.1.

During the last half of 1974 problems with throughput and user controls were uncovered. These problems are documented in last year's annual report [AR 75]. To correct these problems, a short-term project was established to improve these systems. The primary improvements are background queueing and transmission of files, overlapped printing and transmission, and automatic shipment of character sets. The following specific steps were taken:

1. The connection between the PDP-11 and the TIP was changed to use a host interface. Corresponding changes in the software in the PDP-11 were also made.

2. Printing and transmission have been overlapped to achieve maximum throughput

3. Shipment of character sets to the PDP-11 is now performed automatically.

Marti Coale Photos

Figure 7.1 Xerox Graphics Printer and its processor

4. A new foreground program called XGP has been written to replace the old XLIST program. It queues files for printing by a new background process modeled after the LPT server

5. Defaults have been established so that the user has to supply only the name of the file to be printed to the XGP program.

6. A new device, XGP:, has been installed in TENEX so that users may output directly to the XGP

Connection of the PDP-11 as a Host

In order to support high-speed network transmission, the hardware of the PDP-11 has been augmented with a host interface and enough memory to support both the ELF operating system and the XGP program The total core on each PDP-11 is now 64K instead of 32K Memory mapping hardware has also been added.

The old software, based on CMU's PDP-11 XGP program, has been replaced by a combination of VM ELF and MIT's XGP program The VM ELF system provides network and disk I/O and address space management MIT's XGP software is a much improved version of CMU's software, providing the same functions of converting character codes to raster lines suitable for transmission to the XGP hardware

Overlap of Printing with Transmission

Text received from the ARPANET is buffered onto the disk. Printing is initiated after the first page has been received If transmission is slowed after printing has started and the printing process actually catches up to the transmission, printing is interrupted at the next page boundary. Printing resumes when sufficient text has been received.

In normal cases, throughput of a few kilobits per second is all that is required to keep up with the printing process. Even when TENEX is heavily loaded, it is usually able to accomplish this

Automation of Character Set Shipment

The background TENEX process now accepts commands from the text file to ship character sets to the PDP-11. Corresponding changes to XOFF to generate these commands have been made and use of character sets is now controlled entirely by the commands in the text file.

Revision of Core Allocation

The original core allocation scheme in the PDP-11 program placed each new font in progressively increasing memory locations. Eventually, memory space was exhausted and the printing process was aborted. Since only two fonts are active at any one time, it is possible to reuse the space released by previously used fonts. A strategy to reuse the core space was designed and implemented

Establishment of Defaults for XGP and the PDP-11

Defaults for paper size, margins, character sets, and tab stops have been established so that line-printer-type files print as much as possible as they would on the printer.

Queueing of Files

The functions of the original XLIST program have been divided into two parts. One part interacts with the user to accept file names and destinations. It copies the file into the XGP-PRINTER directory. The second part is a permanent background task which attempts to connect to the XGP and sends files stored in the XGP-PRINTER directory to the designated XGP.

Conclusion

This system became operational in late 1975. The XGP special project was disbanded in January 1976 and maintenance of the XGP system is now performed by the ISI Systems Group of which Pete Alfvin is the principal coordinator for the XGP effort.

References

[AR 74] *Annual Technical Report, May 1973 - May 1974*, USC/Information Sciences Institute, ISI-SR-74-1, 1974.

[AR 75] *Annual Technical Report, May 1974 - May 1975*, USC/Information Sciences Institute, ISI-SR-75-3, 1975.

NSW SUPPORT

Research Staff:
Stephen D. Crocker
Chloe Holg
David Wilczynski

The National Software Works (NSW) is an ARPANET-based distributed operating system to provide a uniform computing environment for software developers. Services within the NSW are provided on some of the ARPANET host computers. These computers, called Tool Bearing Hosts, are connected logically by a centralized Works Manager whose responsibilities include maintaining a single NSW file system, validating user rights, and assigning resources. In addition, the NSW includes a user interface, called the Front End, to give the working community access to the NSW's procedures and facilities. (See [Carlson 74], [Crocker 75b], and [Millstein 76], for a deeper view of the NSW.)

Our role in the NSW project was to review technical progress and provide information/consultant service for the project. Our task involved four areas:

- Information and consulting service.
- Participation in the design of the System Design Laboratory (SDL) of the Naval Electronics Laboratory Center (NELC).
- Document control and preparation.
- NSW system testing.

Information and Consulting Service

While the NSW was being designed, we acted as an information source for potential NSW users and other interested parties. In particular, we provided seminars to the U.S. Army Electronics Command at Fort Monmouth, New Jersey, the Naval Air Software Management Advisory Committee (NASMAC) at the Naval Weapons Center at China Lake, and the System Design Laboratory design team at Naval Electronics Laboratory Center in San Diego. We also made presentations at the NBS/ACM/IEEE-sponsored workshop on currently available testing tools in Los Angeles April 1975 [Crocker 75a], at the meeting on Twenty Years of Computer Science in Pisa, Italy during June, 1975 [Crocker 75b], and at the October 1975 meeting of the Los Angeles chapter of Sigspace.

SDL Design with NELC

As an outgrowth of our initial contact with the SDL design team at NELC we were asked to join their design effort to provide NSW expertise after it became clear that NSW was the appropriate vehicle for their system. As active participants, we helped design their initial Operating Capabilities for their first demonstration system. Later we helped write their preliminary design report [NELC 76].

As part of this consulting effort we also helped produce a requirements list for their front-end work station [Wilczynski 76]. That list was communicated to the NSW developers and was acted upon by Stanford Research Institute by means of a change in their front-end specifications.

Document Control and Preparation

ISI has been the documentation center for NSW. Continuing responsibility for documentation will rest with Massachusetts Computer Associates (MCA). Transfer of the activity is in progress. Since early implementations of the NSW were TENEX-based, we received many queries from NSW users about TENEX. To help these users we compiled two documents as an introduction to the TENEX facilities [Holg 76a, Holg 76b].

NSW System Testing

When the NSW became a usable TENEX system, ISI was chosen to house a prototype version for general use. Our system use helped the developers uncover bugs. We also worked directly with the Campus Computer Network people at UCLA to help debug the NSW batch tool interface to their IBM 360 for programs required by the SDL for their IOC system.

Since we were the first "outsiders" to have access to the NSW, we were also the first to have detailed knowledge of how to use the system. Since the NSW user guide was not due until June 1976, we produced one on our own initiative [Holg 76c].

References

[Carlson 74] Carlson, W. E., and S. D. Crocker, "The Impact of Networks on the Software Marketplace," *Proceedings of EASCON 74*, October 1974.

[Crocker 75a] Crocker, S. D., and R. M. Balzer, "The National Software Works: A New Distribution System for Software Development Tools," *Workshop on Currently Available Program Testing Tools*, sponsored by NBS, ACM and IEEE, April 1975

[Crocker 75b] Crocker, S. D., "The National Software Works: A New Method for Providing Software Development Tools Using the ARPANET," *Proceedings from the Meeting on 20 years of Computer Science in Italy*, Pisa, Italy, June 1975 (also *Calcolo*, Vol XII, Supplement 1, 1975)

[Holg 76a] Holg, C. S., *Joy of TENEX*, USC/Information Sciences Institute, Information Memo, April 1976.

[Holg 76b] Holg, C. S., *More Joy of TENEX*, USC/Information Sciences Institute, Information Memo, April 1976.

[Holg 76c] Holg, C. S., and D Wilczynski, *A Very Preliminary NSW User's Guide*, USC/Information Sciences Institute, Information Memo, May 1976.

[Millstein 76] Millstein, R. E., *Semi-Annual Technical Report*, Massachusetts Computer Associates, CADD-7603-0411, March 1976.

[NELC 76] *System Design Laboratory Preliminary Design Report*, NELC TN 3145, March 1976.

[Wilczynski 76] Wilczynski, D., *A Minimum Front-End Proposal*, USC/Information Sciences Institute, Information Memo, February 1976.

TERMINAL DEVELOPMENT

Research Staff:
Robert H. Stotz
Donald Oestreicher
Robert T. Martin
Dono van-Mierop

Background

A significant part of the IA project (Section 5) is devoted to providing good human engineering to the military message service. For the traditional action officer to accept it, the service must be easy and simple to use. This applies particularly to his interaction at the CRT terminal. Therefore, we believe it is critical to provide two-dimensional editing with instantaneous feedback from trivial operations, as though the user's keystrokes actually performed the operation (insert a character, delete a character, move cursor, scroll, iden y a cursor location as being data of interest, etc.). Other more complex operations are dealt with as commands to the system's Agent. For these, indication of command input should be instantaneous, but longer delay in actually performing the action is acceptable.

For a message service test where users are separated by the network from the supporting host computer, it is impossible to supply the necessary speed of response unless processing is done at the terminal site. With this two-stage architecture in mind, the IA project has designed a front end process (called Terminal Control) as a separable module for locally handling actions in the terminal. Terminal Control currently resides in TENEX, but it is being moved to a microprocessor which will then be provided as a part of each CRT terminal on Oahu.

Terminal Control Functions

The Terminal Control manages the terminal keystroke input and display output. On input from the keyboard the Terminal Control does local character buffering, text editing, break determination (when to send the character buffer), local echo to the display, and text editing (on the input character buffer or the output display buffer), etc. On output to the terminal it handles multiple windows (contiguous areas of screen) and domains (logical blocks of text within windows), dynamic formatting of data in screen windows, buffering of more data than can be displayed in a window, scrolling of windows, maintenance of a logical cursor address, and interface to the rest of the system. Domains have properties assigned them by their owner (whatever module created the display) such as highlight, editable, break characters allowed, etc., which allows the Terminal Control to handle keystroke input differently in different domains.

The Terminal Control is general-purpose in nature, so that it can be effective for other message services for this test and for applications beyond military message handling. With appropriate host software, it will support a variety of styles of command language input (teletype style, command window style in the manner of NLS, menu selection style, etc.) as well as two-dimensional full screen editing. An additional function that must be provided at the terminal for the Military Message Experiment on Oahu is a piece of hardware that indicates the security level at which the user is operating and that provides several keys for confirming or not confirming changes to his level. This equipment, called the Security Control Box, must be "trusted" in a security sense and is therefore not to be incorporated into the terminal microprocessor.

Why These Functions Should Be in the Terminal

For several reasons, we believe the terminal is the right location for this processing. First, good response is guaranteed. The blocked nature of the data transfers will reduce network and TENEX overhead. It also provides maximum configuration flexibility, since stand-alone terminals can work through TIPs or data concentrators at any point on the network. Last, it is consistent with the security approach being adopted for the Oahu test, and for end-to-end encryption, should that ever be added to the system. The trend is toward more capacity per dollar invested in each terminal, so that even if this approach is not the most cost-effective now, we are confident it will be within two or three years.

Terminal Selection

A study of available terminals failed to locate one with the desired display characteristics, and a 64K byte address space, and running at 1 to 10 microseconds per instruction; we have therefore decided to use a new upgraded version of the HP 2640A terminal, which contains an 8080 microprocessor in place of the 8008. The 2640A was chosen by ISI a year ago for in-house terminal use because it offered the best design and reliability available at that time. In general, ISI has been pleased with the performance of these terminals.

Plan

The basic approach for early development of terminal software is to emulate the display processor in PRIM (Section 2) and develop the bulk of our software there, which has the advantage of having available the powerful creation and debugging tools of the PRIM environment. The major shortcoming of this approach is that PRIM has virtually no I/O, which is the terminal microprocessor's major activity. Thus, though we can determine that the individual programs do what we expect them to do, we cannot check out the real-time aspects of the integrated program operation. This step will be done on a prototype terminal which contains all (64K) read/write memory (RAM).

Besides the major task of putting the Terminal Control into the terminal microprocessor, some minor hardware changes are planned. For example, it is necessary to label the key caps of all the function keys with appropriate mnemonics and to add external circuitry to drive the Security Control Box.

The plan calls for four terminals equipped with 64K of RAM memory for development and checkout work, then "production" terminals for the operational test equipped with the proper mix of ROM (program store) and RAM (data and buffer store). We have estimated this split to be 16K RAM and 48K ROM. One prototype terminal is required for initial development and debugging. Three more prototypes will then be produced and given to the message service developers (ISI, BBN, and MIT). After the prototype units have been shaken down with the message services, "production" terminals will be made for the operational test, a step which involves producing 24 ROM masks at a high one-time cost (any changes in the firmware after these ROMs have been made will be equally costly). Production terminals for the test will be delivered together with wheeled tables with folding wings.

The Security Control Box will be provided with approximately four LEDs to indicate the level of security of the screen.

Progress to Date

A specification for the external characteristics of the terminal has been submitted to the participants in the Military Message Experiment. Internal program specifications have been written and programming begun. A PRIM 8080 emulator is operating and several routines have been written and checked out on it. ISI is using a prototype of the new HP terminal for evaluation and initial program development; this will be replaced by an early production unit, to be purchased when available.

8.
ARPANET TENEX SERVICE

Technical Staff:
Marion McKinley, Jr.
Pete Alfvin
Alan E. Algustyniak
Dale Chase
George Dietrich
Glen W. Gauthier
Donald R. Lovelace
Raymond L. Mason
William H Moore
Donna J Nagel
Clarence Perkins
Vernon W Reynolds
Dale S. Russell
Leo Yamanaka

Support Staff:
Ralph W. Caldwell
Wanda N. Canillas
Larry Fye
Oralia E. Garza
Delia A. Heilig
Kyle P. Lemmons
James W. McKinley
Gary Seaton
Rennie Simpson
Michael E Vilain
Deborah C. Williams

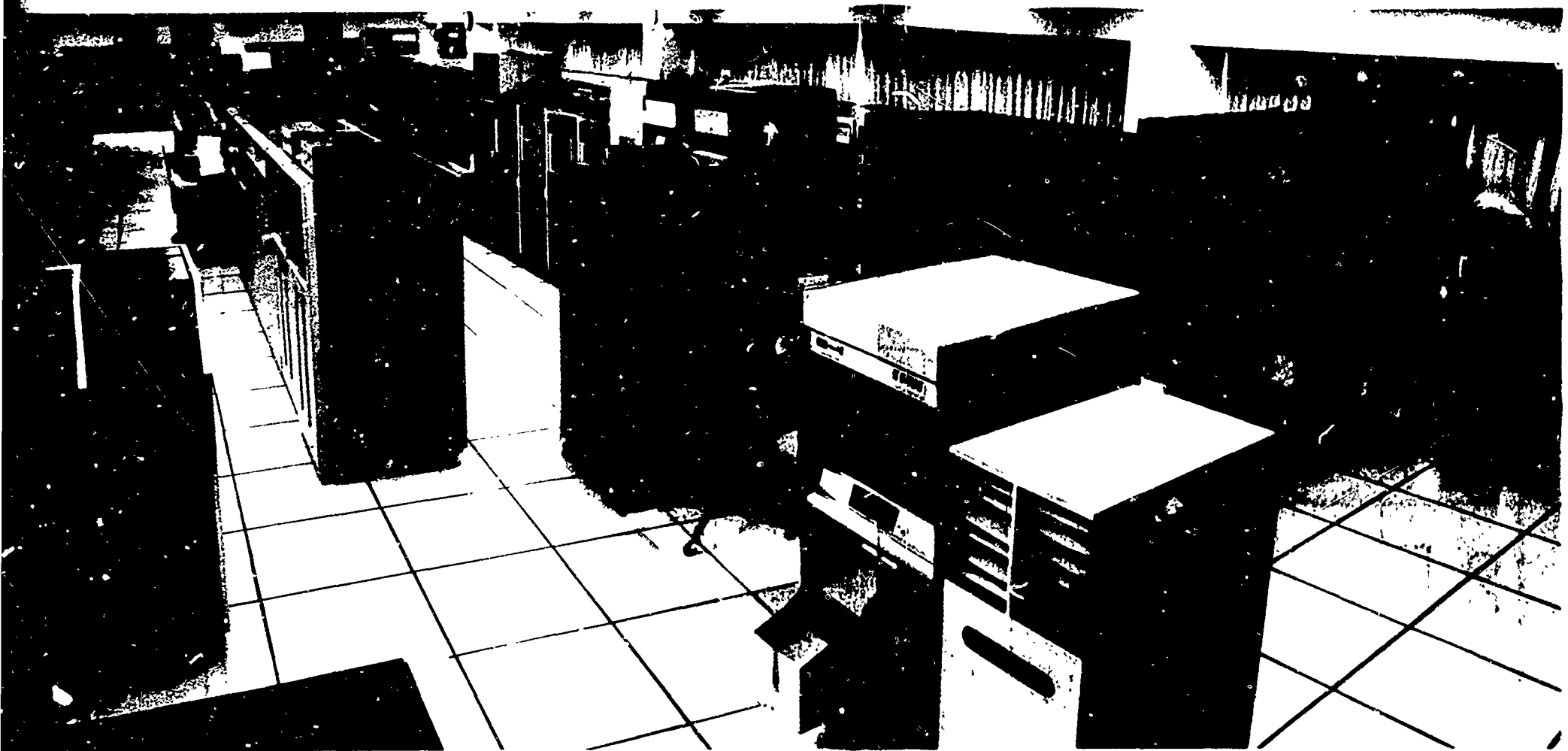
INTRODUCTION

The ISI ARPANET TENEX service facility is operated as a development and service center in support of a broad set of ARPA projects. It currently services more than 1000 directories, some of which are multiplexed by several users. Approximately 95 percent of the users access the facilities via the ARPANET from locations extending from Europe to Hawaii. All of the facility's systems are available to all users, whether they are connected through the ARPANET locally or remotely.

The facility consists of five Digital Equipment Corporation central processors (one KI-10, one KL-2040, and three KA-10s), Bolt Beranek and Newman virtual memory paging boxes, large-capacity memories, on-line swapping and file storage, and associated peripherals (see Figures 8.1 and 8.2). All systems presently run under control of the TENEX or TOPS-20 operating system (originally developed by Bolt Beranek and Newman), which supports a wide variety of simultaneous users.

HARDWARE

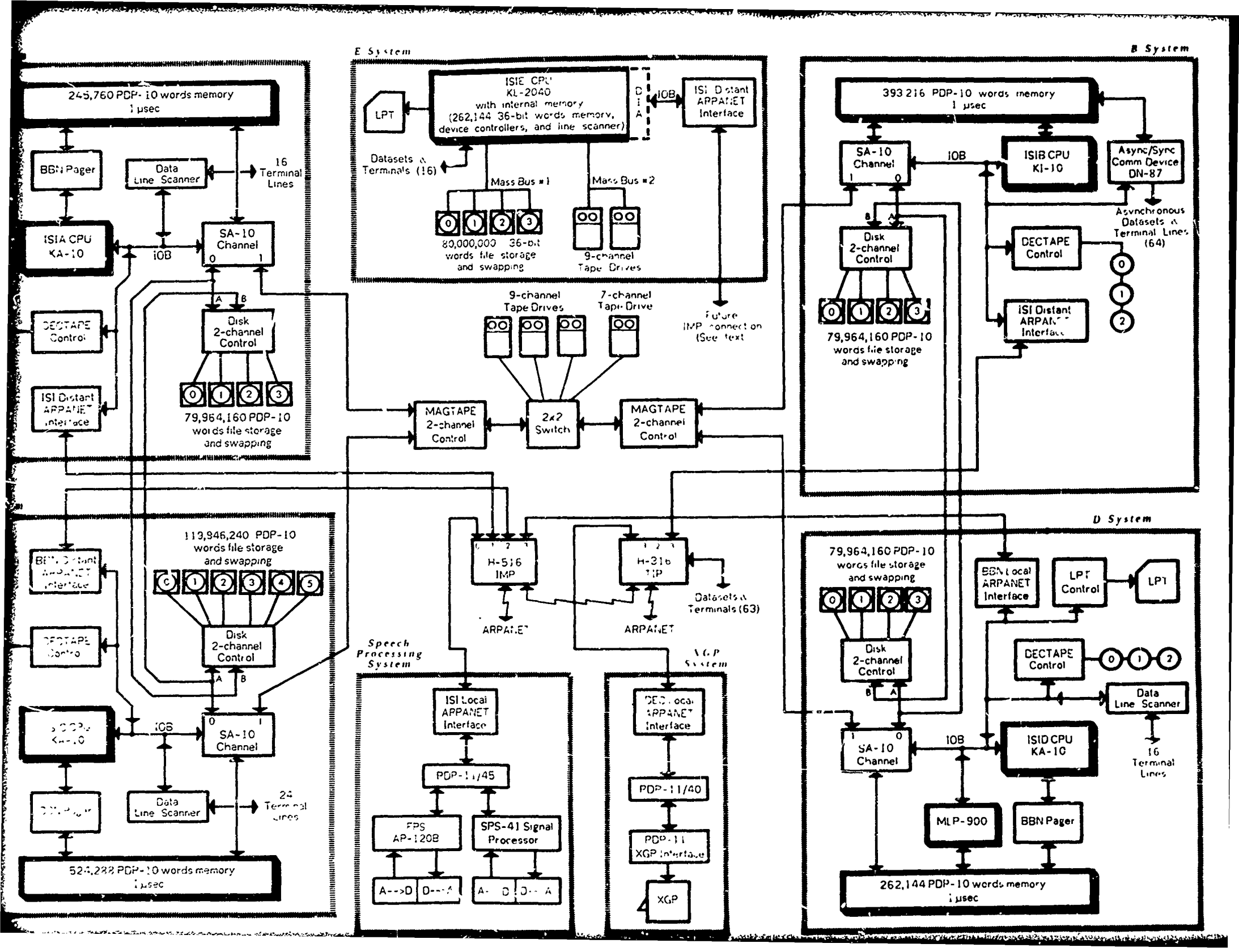
New hardware acquired in the past year includes one additional DEC KL-2040 (TOPS-20) system with 256K words of DEC high-speed internal memory and associated peripheral devices (Figure 8.3), two CALCOMP 230 disk drives presently attached to ISIC for additional on-line swapping and file storage capabilities, and a new CALCOMP 347 magnetic tape drive which is shared (as are all ISI magnetic tape facilities) among four TENEX systems. Figure 8.2 shows the current ISI facility configuration. Note that none of the central processors (KA-10s, KL-2040, or KI-10) operate in dual processor mode.



Marti Coale Photos

Figure 8.1 Composite photograph of computer room

Figure 8.1 Composite photograph of computer room


E System
B System
D System
Speech Processing System
XGP System
245,760 PDP-10 words memory
1 μsec
BBN Pager
Data Line Scanner
16 Terminal Lines
ISIA CPU KA-10
IOB
SA-10 Channel
DECTAPE Control
ISI Distant ARPANET Interface
Disk 2-channel Control
79,964,160 PDP-10 words file storage and swapping
ISIE CPU KL-2040 with internal memory (262,144 36-bit words memory, device controllers, and line scanner)
LPT
DIA
IOB
ISI Distant ARPANET Interface
Datasets & Terminals (16)
Mass Bus #1
Mass Bus #2
80,000,000 36-bit words file storage and swapping
9-channel Tape Drives
9-channel Tape Drives
7-channel Tape Drive
Future IMP connection (See text
MAGTAPE 2-channel Control
2x2 Switch
MAGTAPE 2-channel Control
393,216 PDP-10 words memory
1 μsec
SA-10 Channel
IOB
ISIB CPU KI-10
Async/Sync Comm Device DN-87
Asynchronous Datasets & Terminal Lines (64)
Disk 2-channel Control
DECTAPE Control
ISI Distant ARPANET Interface
79,964,160 PDP-10 words file storage and swapping
119,946,240 PDP-10 words file storage and swapping
Disk 2-channel Control
SA-10 Channel
IOB
Data Line Scanner
24 Terminal Lines
524,288 PDP-10 words memory
1 μsec
H-516 IMP
H-316 TIP
ARPANET
ARPANET
Datasets & Terminals (63)
ISI Local ARPANET Interface
PDP-11/45
FPS AP-120B
SPS-41 Signal Processor
DEC Local ARPANET Interface
PDP-11/40
PDP-11 XGP Interface
XGP
79,964,160 PDP-10 words file storage and swapping
Disk 2-channel Control
SA-10 Channel
IOB
ISID CPU KA-10
BBN Local ARPANET Interface
LPT Control
LPT
DECTAPE Control
Data Line Scanner
16 Terminal Lines
MLP-900
BBN Pager
262,144 PDP-10 words memory
1 μsec

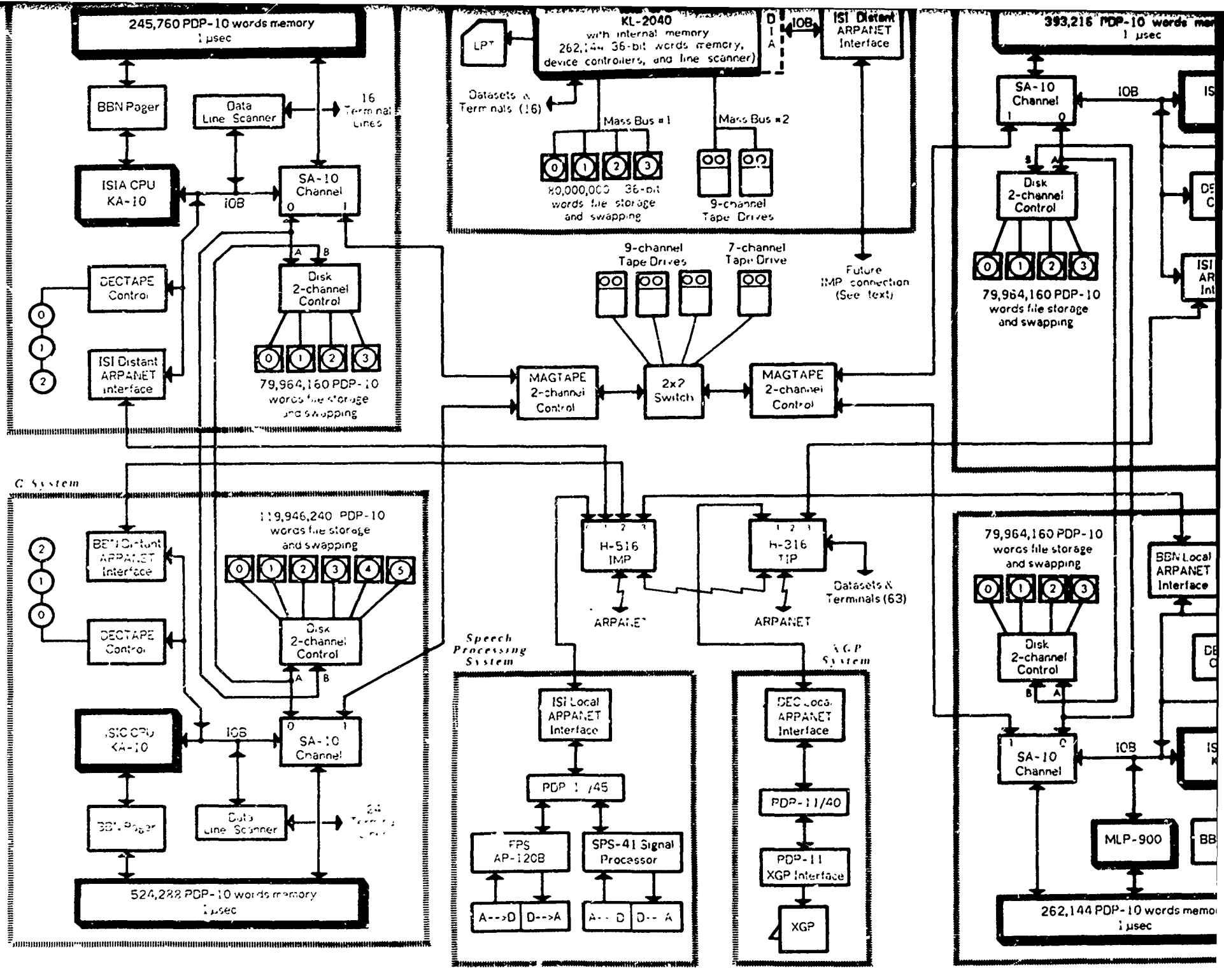
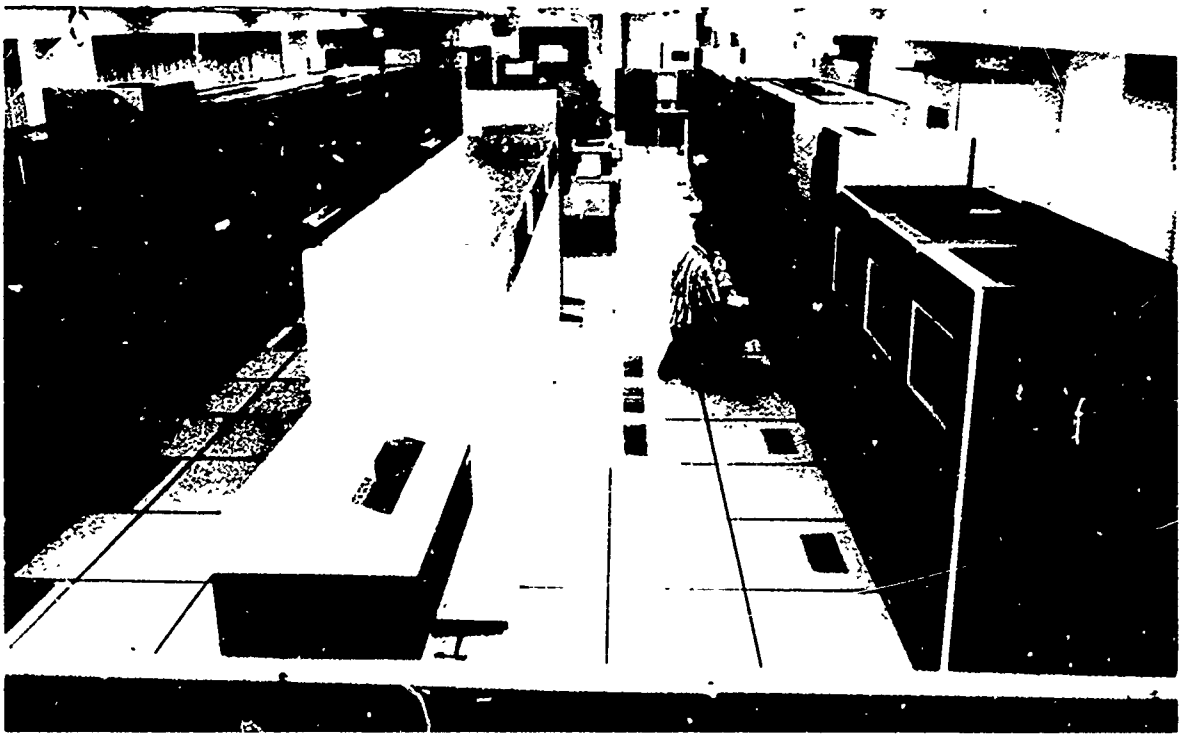


Figure 8.2 Diagram of ISI ARPANET TENEX service facility



Marti Coale Photos

Figure 8.3 View of computer room with KL-2040 in center foreground.

Instead, the main goal of having the several systems is to provide a significant increase in the availability of the ISI primary machines: Thus if one of the systems designated as a primary machine crashes or is unavailable because of hardware/software maintenance or development, then one of the other systems may be started as a primary machine and service continued after a brief (15 minutes or less) interruption to switch the file storage media and one cable.

Also included within the TENEX service facility are one BBN H-516 Interface Message Processor (IMP), one BBN H-316 Terminal Interface Processor (TIP), one DEC PDP-11/40 and Xerox Graphics Printer (XGP), one DEC PDP-11/45 with an SPS-41 Signal Processing System and a Floating Point Systems AP-120B (FBS) (configured as a speech processor), one microprogrammable processor (MLP-900) and several associated peripheral devices such as disk, memory, special ISI-developed interfaces, TTY's etc.

Purchase orders have been submitted for additional hardware, a DEC DN87 Universal Synchronous/Asynchronous Front End Communications system that will allow all in-house users to directly access the in-house (ISIB KI-10) TENEX system. This will eliminate the requirements for the BBN-316 TIP at ISI. Delivery of this communications equipment is expected prior to July 31, 1976. Thirty days after delivery the ISI TIP will be replaced by a BBN IMP with four host ports, which will allow ISI to accommodate two additional host systems that will allow additional users to access future ISI systems via the ARPANET.

SOFTWARE

During the year a concentrated effort has been made to ensure that all of the ISI TENEX service machines provided the same level of systems software. TENEX version 1.34 with some necessary local modifications was installed, and the most recent subsystems and documentation packages were obtained and released. To aid in this continuing effort of software maintenance and upgrading, a method for automatically distributing changes among all ISI service machines is now in the developmental stage and will be implemented in the near future. We have also provided load-leveling across the machines in conjunction with ARPA/IPTO to assure reasonable response and greatly expanded system utilization.

Two major services, XGP printing and file archival, were improved. The XGP driver software running under TENEX and on the PDP-11 was upgraded and stabilized, XOFF problems were corrected, and special output options and software interfaces were implemented (see Section 7 for details). This effort was also applied to the ARPA XGP to bring it to the same level of service, which facilitates system maintenance. File archival changes were necessitated by incompatibilities between old magtape drives and the new CALCOMP magtape system. The old archive library was copied into a new, substantially smaller library, reflecting the most recent format changes; this made the entire library accessible to all ISI TENEX service machines.

Several members of the software group have been actively engaged in performing comparisons of the TENEX operating systems with the TOPS-20 operating system. Upon acceptance of the TOPS-20 system at ISI, maximum effort will be devoted to installation of new JSYS's, JSYS traps, features, and modifications that will allow the majority of the existing TENEX subsystems to operate under the TOPS-20 system. Modifications of many of the existing TENEX subsystems will also be required as part of this effort. Attempts will also be made to take the existing ARPA Network Control Protocol in TENEX and, with appropriate modifications, incorporate it into the TOPS-20 operating system. This, along with a hardware network interface developed, assembled, and integrated into the KL-2040 hardware by ISI, will allow the TOPS-20 system to be accessed via the ARPANET.

SUPPORT PERSONNEL

ISI provides seven-day-a-week, twenty-four-hour-a-day operator, software, and hardware support for the TENEX service facility. At least one operator is physically on-site at all times, and the systems programmers and computer service engineers either are physically on-site or are scheduled for one-hour on-call service.

RELIABILITY

To provide required hardware/software preventive and/or corrective maintenance of the equipment, ISI will continue scheduling each of the TENEX systems as "out of service" (unavailable to users) for seven contiguous hours each week. The remaining 161 hours of each week are intended to be devoted entirely (100 percent) to user service. The actual long-term up-time for the network service machine has exceeded 99 percent of scheduled up-time for the last year.

LOCAL PROJECT SUPPORT

The TENEX facility has been used extensively in support of local projects. The ISI staff makes use of the available standard subsystems (e.g., editors, compilers, assemblers, and utilities), and some staff members have written subsystems and utilities to support their own projects. The facility also supports less frequently used subsystems at the special request of users (e.g., PDP-11 cross-assemblers and the DECUS Scientific Subroutine Package).

Major TENEX monitor modifications and a new software driver package for support of the MLP-900 (microprogrammable processor originally developed for the PRIM project) have been developed and verified and are now operational on ISID (see Section 2 for details). These modifications allow more efficient use of the processor-to-processor communication facilities between the TENEX operating system and the MLP-900.

PUBLICATIONS

Research Reports

Abbott, Russell J., *A Command Language Processor for Flexible Interface Design*, ISI/RR-74-24, February 1975.

Anderson, Robert H., *Programmable Automation: The Future of Computers in Manufacturing*, ISI/RR-73-2, March 1973; also appeared in *Datamation*, Vol. 18, No. 12, December 1972, pp. 46-52.

---, and Nake M. Kamrany, *Advanced Computer-Based Manufacturing Systems for Defense Needs*, ISI/RR-73-10, September 1973.

Balzer, Robert M., *Automatic Programming*, ISI/RR-73-1 (draft only).

---, *Human Use of World Knowledge*, ISI/RR-73-7, March 1974.

---, *Imprecise Program Specification*, ISI/RR-75-36, May 1976; also appeared in *Calcolo*, Vol. XII, Supplement 1, 1975.

---, *Language-Independent Programmer's Interface*, ISI/RR-73-15, March 1974; also appeared in *AFIPS Conference Proceedings*, Vol. 43, AFIPS Press, Montvale, N. J., 1974.

---, Norton R. Greenfeld, Martin J. Kay, William C. Mann, Walter R. Ryder, David Wilczynski, and Albert L. Zobrist, *Domain-Independent Automatic Programming*, ISI/RR-73-14, March 1974; also appeared in *Proceedings of the International Federation of Information Processing Congress*, 1974.

Bisbey, Richard L., Jim Carlstedt, Dale M. Chase, and Dennis Hollingworth, *Data Dependency Analysis*, ISI/RR-76-45, February 1976.

---, and Gerald J. Popek, *Encapsulation: An Approach to Operating System Security*, ISI/RR-73-17, December 1973.

Carlisle, James H., *A Tutorial for Use of the TENEX Electronic Notebook-Conference (TEN-C) System on the ARPANET*, ISI/RR-75-38, September 1975.

Carlstedt, Jim, Richard L. Bisbey II, and Gerald J. Popek, *Pattern-Directed Protection Evaluation*, ISI/RR-75-31, June 1975.

Cohen, Dan, *Specification for the Network Voice Protocol*, ISI/RR-75-39, March 1976.

Ellis, Thomas O., Louis Gallenson, John F. Heafner, and John T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, ISI/RR-73-12, June 1973.

Gallenson, Louis, *An Approach to Providing a User Interface for Military Computer-Aided Instruction in 1980*, ISI/RR-75-43, December 1975.

Good, Donald I., Ralph L. London, and W. W. Bledsoe, *An Interactive Program Verification System*, ISI/RR-74-22, November 1974; also appeared in *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 59-67.

Heafner, John F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, ISI/RR-74-21, September 1974.

---, *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, ISI/RR-75-34, July 1975.

Igarashi, Shigeru, Ralph L. London, and David C. Luckham, *Automatic Program Verification I: A Logical Basis and Its Implementation*, ISI/RR-73-11, May 1973; also appeared in *Artificial Intelligence Memo 200*, Stanford University, May 1973 and *Acta Informatica*, Vol. 4, No. 2, 1975, pp. 145-182.

Kamrany, Nake M., *A Preliminary Analysis of the Economic Impact of Programmable Automation Upon Discrete Manufacturing Products*, ISI/RR-73-4, October 1973.

Mann, William C., *Dialogue-Based Research in Man-Machine Communication*, ISI/RR-75-41, November 1975.

Martin, Thomas H., Monty C. Stanford, F. Roy Carlson, and William C. Mann, *A Policy Assessment of Priorities and Functional Needs for the Military Computer-Aided Instruction Terminal*, ISI/RR-75-44, December 1975.

Oestreicher, Donald R., *A Microprogramming Language for the MLP-900*, ISI/RR-73-8, June 1973; also appeared in the Proceedings of the ACM Sigplan Sigmicro Interface Meeting, New York, May 30-June 1, 1973.

Richardson, Leroy, *PRIM Overview*, ISI/RR-74-19, February 1974.

Rothenberg, Jeff, *An Editor to Support Military Message Processing Personnel*, ISI/RR-74-27, June 1975.

---, *An Intelligent Tutor: On-Line Documentation and Help for A Military Message Service*, ISI/RR-74-26, May 1975.

Tugender, Ronald, and Donald R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, ISI/RR-74-23, May 1975.

Wilczynski, David, *A Process Elaboration Formalism for Writing and Analyzing Programs*, ISI/RR-75-35, October 1975.

Yonke, Martin D., *A Knowledgeable, Language-Independent System for Program Construction and Modification*, ISI/RR-75-42, December 1975.

Special Reports

Annual Technical Report May 1972 - May 1973, ISI/SR-73-1, September 1973.

A Research Program in the Field of Computer Technology, Annual Technical Report, May 1973 - May 1974, ISI/SR-74-2, July 1974.

A Research Program in Computer Technology, Annual Technical Report, May 1974 - June 1975, ISI/SR-75-3, September 1975.

Bisbey, Richard L., Gerald Popek, and Jim Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time*, ISI/SR-75-4, January 1976.

Carlstedt, Jim, *Protection Errors in Operating Systems: Validation of Critical Variables*, ISI/SR-75-5, May 1976.

Hollingworth, Dennis, and Richard L. Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals*, ISI/SR-76-7, June 1976.

Technical Manuals

Gallenson, Louis, Joel Goldberg, Ray Mason, Donald Oestreicher, and Leroy Richardson, *PRIM User's Manual*, ISI/TM-75-1, May 1975.

XED Manual, ISI/TM-76-3, May 1976.